VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Metody komprese bioinformatických dat pro přenos na HPC infrastrukturu

Data Compression Methods for Bioinformatic Data Transfer on HPC Infrastructure

2020 Bc. Vojtěch Moravec

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Vojtěch Moravec**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Metody komprese bioinformatických dat pro přenos na HPC infrastrukturu
Data Compression Methods for Bioinformatic Data Transfer on HPC Infrastructure

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je prostudovat možné metody komprese obrazových dat a navrhnout možnosti jejich využití v projektu Bioimageinformatics on HPC, který se zabývá přenosem a následným zpracováváním velkých objemů bioinformatických dat na HPC infrastruktuře. Student se zaměří na srovnání dostupných metod pro kompresi obrazových dat a integraci těch nejvhodnějších (dle výsledků jejich srovnání) do aplikačního rámce HEAppE (HPC as a Service), který řeší vzdálený přístup k HPC infrastruktuře a řídí samotný přenos dat pro zpracování na HPC. V práci se zaměřte na:
1. Přehled současného stavu poznání - popis zpracovávaných bioinformstických dat, aktuálně využívané metody komprese v této oblasti
2. Návrh dalších využitelných metod komprese dat, či úpravy staávajících metod
3. Experimenty a jejich následné vyhodnocení.

Seznam doporučené odborné literatury:

[1] Salomon, David. Data Compression: The Complete Reference. 4th ed. London: Springer-Verlag, 2007. ISBN 1846286026.
[2] Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Mgr. Jiří Dvorský, Ph.D.**

Datum zadání: 01.09.2019
Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 15. 5. 2020

Moravec Vojtěch

Rád bych na tomto místě poděkoval svému vedoucímu diplomové práce doc. Mgr. Jiřímu Dvorskému, Ph.D., za vstřícný přístup, ochotu a čas věnovaný k pomoci na této práci. Dále bych chtěl poděkovat Mgr. Ing. Michalu Krumniklovi, Ph.D. za jeho rady a ochotu. V neposlední řadě bych rád poděkoval své přítelkyni a rodině za podporu.

Abstrakt

Bioinformatická data patří v současné době k jedním z nejdůležitějších typů dat pro klinický a biologický výzkum. S rychlým vývojem mikroskopických technologií je kvalita a velikost datasetů stále větší. Cílem této práce je prozkoumat metody komprese bioinformatických dat. Komprese by zajistila rychlejší přenos a menší nároky na uchovávání těchto datasetů. Nejprve zběžně uvedeme existující metody bezztrátové komprese a dále se budeme detailněji zabývat kompresi ztrátovou. Konkrétně se bude jednat o algoritmy skalární a vektorové kvantizace. Tyto metody budou vyzkoušeny na reálných datech. Součástí této práce jsou výsledky zkoumání, spolu s detailní analýzou kvality ztrátové komprese a vzniklé kompresní chyby. Výsledky dokazují, že pomocí navrhnutých algoritmů, lze na testovaných datech dosáhnout dobrých kompresních výsledků. Testovaný dataset vykazuje minimálně znatelnou chybu v obraze, při velmi dobrých kompresních poměrech.

Klíčová slova: bioinformatická data, komprese obrazu, ztrátová komprese, skalární kvantizace, vektorová kvantizace

Abstract

Bioinformatical data are currently one of the most valuable data types for clinical and biological research. The fast development of microscopy techniques leads to more data with higher quality than ever before. This thesis aims to explore methods of compression of bioinformatical data. The compression would result in faster transmission times and smaller requirements for data storage. First, we describe the existing methods of lossless data compression and then we take a closer look at the lossy compression techniques. Specifically, the scalar quantization and vector quantization algorithms. These methods will be tested on real-life data. Results of compression, with a detailed analysis of a lossy compression results and the compression error, will be presented in this work. Obtained results, prove that our algorithms can achieve very good compression for the tested data. We can keep most of the image quality while minimizing the compression ratio.

Keywords: bioinformatical data, image compression, lossy compression, scalar quantization, vector quantization

Obsah

Seznam použitých zkratek a symbolů

BPP	–	Počet bitů na pixel
CR	–	Kompresní poměr
MAE	–	Průměrná absolutní chyba
MSE	–	Průměrná kvadratická chyba
PSNR	–	Špičkový poměr signálu k šumu
QCMP	–	Vlastní formát komprimovaného souboru *Quantization Compressed File*

Seznam obrázků

Seznam tabulek

1 Úvod

Bioinformatická data patří v současné době k jedněm z nejdůležitějších typů dat pro klinický a biologický výzkum. Rychlý vývoj v kvalitě mikroskopických technologií a metod, nebo nalezení zcela nových způsobů, jak pořizovat tato data, vede k možnosti získávat obrovské množství biologických dat ve vyšší kvalitě než kdy předtím. V rámci této práce se budeme zabývat obrazovými daty, u kterých je vyžadována co největší kvalita. Setkáváme se tedy například s 16 či 32 bity na pixel, oproti standardním 8 bitům. Větší počet bitů zajišťuje širší rozsah hodnot jednotlivých pixelů, ale znamená také větší velikost souboru.

Velké datasety, získané pomocí mikroskopů, jsou dále zpracovávány na superpočítačích. Zde jsou často pomocí algoritmů zpracování obrazu převedeny do sofistikovanějších reprezentací. Jako příklad těchto algoritmů můžeme uvést redukci dimenze, vlnkovou transformaci nebo reprezentaci pomocí obrazových pyramid. Pro zpracování je potřeba, aby byla data přenesena přes síť do uložiště superpočítače. Existují nástroje jako je plugin BigDataViewer [1] do programu Fiji [2], které nám dovolují s těmito daty pracovat. Tento plugin dovoluje prohlížet velké obrazové datasety přes internet, a to díky vykreslování na straně klienta a chytrého využití mezipaměti na straně serveru. Přes internet je tedy přenesena pouze ta část datasetu, která ještě nebyla přenesena dříve. Avšak i tak je mezi klientem a serverem přeneseno velké množství dat, často v řádech stovek megabytů až jednotek gigabytů.

Z těchto důvodů dává smysl uvažovat o kompresi bioinformatických dat, která by vedla ke zmenšení jejich velikosti, zrychlení přenosu mezi počítači a snížila by nároky na jejich ukládání. V první části této práce, uvedeme několik vybraných metod bezztrátové komprese, které vyzkoušíme na CZI [3] souborech. Následně se budeme detailněji zabývat ztrátovou kompresí. Ztrátová komprese nám dovoluje dosahovat mnohem lepších kompresních poměrů, za cenu ztráty určité obrazové informace. Cílem je tedy dosáhnout co nejlepšího kompresního poměru a zachovat co nejvíce kvality. Chyba vzniklá kompresí musí být minimální. Rychlost komprese a dekomprese je pro nás také důležitá a měla by být co nejvyšší. Při slibných výsledcích, bychom chtěli zakomponovat naše metody do pluginu BigDataViewer.

Námi navrhnuté ztrátové kompresní algoritmy jsou založeny na technikách skalární a vektorové kvantizace. Jednoduší algoritmus, skalární kvantizace, funguje pomocí redukce všech možných hodnot pixelu na vybrané slovníkové hodnoty. Jelikož je množství slovníkových hodnot omezeno je tento algoritmus ztrátový. Oproti skalární kvantizaci, která bere v potaz pouze jednotlivý pixel, vektorová kvantizace využívá podobnosti sousedních pixelů. Pomocí několika sousedících pixelů vytváří slovník vícedimenzionálních objektů. Pro standardní obrazy můžeme těmito objekty rozumět řádkové vektory či dvoudimenzionální maticové vektory. Light-sheet mikroskopie v základu produkuje 3D volumetrická data, které se dají prohlížet v prostoru, a proto můžeme také využit tensorů jako objektů ve slovníku.

Tyto vybrané metody budou vyzkoušeny na reálných datech a bude uvedena jejich detailní analýza. Součástí této analýzy bude prozkoumání kvality kompresních metod, jakého dosahují

kompresního poměru a analýza chyby, která vznikla kompresí. V rámci této práce bude vytvořen nástroj, který dovoluje provést kompresi i dekompresi 16 bitových obrazů.

Pokud se ukáže, že vybrané algoritmy fungují dobře a kompresní chyba je vizuálně zanedbatelná, chtěli bychom napojit naši kompresi do Server-Klient aplikace BigDataVieweru. Na straně serveru by to znamenalo přidání kompresní vrstvy. Tato vrstva by se starala o převod originálních dat do komprimované podoby. Klient by byl modifikován tak, že by byl schopen přijmout komprimovaný proud dat, který by dekomprimoval zpět na vykreslovaný obraz. Toto vylepšení BigDataViewer prohlížeče by eventuálně vedlo k responsivnějšímu prohlížení, bez velké ztráty důležité obrazové informace. Jelikož zpracování bioinformatických datasetů obvykle probíhá na superpočítačích, mohou být přidané časové nároky na kompresi zanedbány. Komprese dat může být například zahrnuta jako jeden z posledních kroků analýzy a zpracování. Dekomprese je již oproti kompresi nenáročná a může být provedena na straně klienta.

2 Bezztrátová komprese obrazu

Jak již bylo zmíněno v úvodu této práce, v první sekci se budeme zběžně zabývat využitím existujících metod komprese. Bude se jednat hlavně o bezztrátovou kompresi obrazu. Tato část práce byla vytvořena v rámci semestrálního projektu. Vyzkoušíme zde několik bezztrátových algoritmů komprese a také jeden algoritmus, který byl vytvořen přímo pro kompresi biologických obrazů. Tento speciální algoritmus nabízí také ztrátovou kompresi. Cílem této části bylo vyzkoušet existující algoritmy na datech, které jsou přenášeny v CZI souborech.

2.1 ZISRAW (CZI) formát

Údaje uvedené v této sekce vychází z oficiální specifikace formátu CZI [3], společnosti Carl Zeiss ZEN software. Formát CZI byl vytvořen pro ukládání výsledku mikroskopických metod, tedy obrazů, vytvořených pomocí mikroskopů. Při jeho návrhu bylo také myšleno na metadata, která popisují jak samotnou obrazovou informaci, tak i postup jejího získání. Tímto postupem rozumíme podmínky, za kterých byl obraz pořízen. Jako příklad můžeme uvést zvolenou metodu mikroskopie, způsob nasvětlení vzorku, úhel snímání atd. Tato metadata jsou ukládána ve formátu XML s kódováním UTF-8 a vychází ze specifikace OME (*Open Microscopy Environment*) [4].

CZI formát vychází z ještě obecnějšího formátu ZISRAW, jehož hlavním cílem je umožnit přenos velkého množství dat v binární podobě. S tímto úmyslem byla navrhnutá struktura tohoto formátu, který se skládá z jednotlivých segmentů. Každý segment je identifikován svou hlavičkou, která obsahuje informace o bloku dat, který se nachází za ní. Navíc, pro rychlejších navigaci v souboru existují speciální adresářové segmenty, které odkazují na polohu jednotlivých segmentů v souboru.

Samotná obrazová informace, která je pro nás zajímavá, se nachází v segmentech, označených jako ZISRAWSUBBLOCK. V tomto segmentu dále nalezneme metadata specifická pro daný snímek, jako je například počet bitů na pixel, čas pořízení anebo přesná pozice mikroskopu. Co se týče formátu obrazu, tak CZI soubory dovolují použití několika typu pixelů, jak pro barevné, tak i jednokanálové (černobílé) obrazy. Seznam všech podporovaných typů pixelu najdeme v Tabulce 1. Obecně nejpoužívanější typem pixelu je Gray16, kde jeden pixel je uložen pomocí 16 bitů a rozsah jeho hodnoty je od 0 do 65535. V této práci se zaměříme právě na obrazy s tímto typem pixelu.

Velikost bioinformatického datasetu je většinou dána počtem snímků, který je v něm obsažen. Často je biologický vzorek snímán v čase nebo s měnící se polohou mikroskopu. Pořizování obrazů tímto způsobem produkuje velké množství dat. Velké velikosti datasetů jsou tedy spíše způsobeny počtem snímků, které obsahují nežli rozlišením jednotlivého snímku. Navíc s využitím více bitů na pixel velikost datasetu roste. V jakém čase a z jaké pozice mikroskopu byl snímek pořízen, se dozvíme v jeho specifických metadatech. Dále zde najdeme informace o jeho rozměrech.

Typ	Bitů na pixel	Typ čísla
Gray8	8	Celočíselné
Gray16	16	Celočíselné
Gray32Float	32	Desetinné
Bgr24	24	Celočíselné
Bgr48	48	Celočíselné
Bgr96Float	96	Desetinné
Bgra32	32	Celočíselné
Gray64ComplexFloat	64	Desetinné
Bgr192ComplexFloat	192	Desetinné
Gray32	32	Celočíselné
Gray64	64	Celočíselné

Tabulka 1: Podporované typy pixelů v CZI souborech

Často se bavíme o řezech obrazu, neboť vzorek je postupně snímán po celé jeho délce, jako je tomu například u počítačové tomografie.

Někdy se může stát, ačkoli je typ pixelu Gray16, že bude barevný obraz rozdělen do několika kanálů, které budou znovu spojeny až při nebo po analýze. Příklad tohoto můžeme vidět na Obrázku 1.

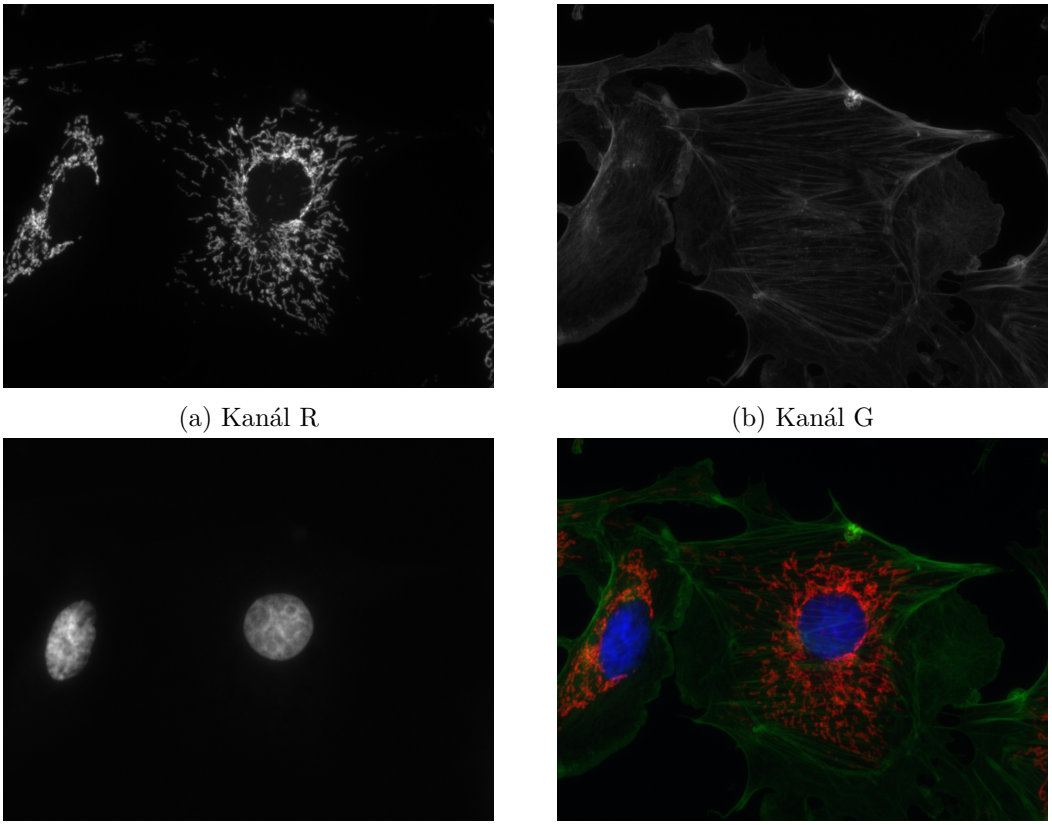
Formát CZI v základu podporuje tři různé kompresní algoritmy pro kompresi obrazových dat. Nutno podotknout, že u dat, které jsme měli k dispozici, jsme se většinou setkali s úplnou absencí komprese. Specifikací je povolen slovníkový kompresní algoritmus LZW a dvě varianty JPEG komprese, standardní JPEG a JPEG-XR. JPEG-XR je modernější variantou a nabízí lepší kompresní poměry spolu s bezztrátovou kompresí. Více o JPEG kompresi se čtenář dozví v kapitole 3.1.

2.2 Vybrané bezztrátové kompresní algoritmy

V této kapitole velice zjednodušeně uvedeme vybrané kompresní algoritmy, které jsme vyzkoušeli na snímcích z CZI souborů. Nejprve představíme tři algoritmy, které se používají pro bezztrátovou kompresi dat obecně, a staly se *de facto* standardem. Dále popíšeme algoritmus B^3D [5], který se přímo zaměřuje na kompresi snímků získaných z mikroskopu. V neposlední řadě vyzkoušíme zakódovat snímek jako rozdíl od předcházejícího snímku nebo metodu transformace dat.

V celé této práci, budeme hodnotit kompresní algoritmy pomocí kompresního poměru CR, který je definován podle Rovnice 1.

$$\mathrm{CR} = \frac{\text{Počet bytů po kompresi}}{\text{Počet bytů před kompresí}} \tag{1}$$



(a) Kanál R (b) Kanál G

(c) Kanál B (d) Složený obraz

Obrázek 1: Ukázka kompozice tří Gray16 snímků do barevného obrazu

Cílem je tedy dosáhnout co nejmenších hodnot CR, menších jak $1,0$. Jestliže by byla velikost souboru po kompresi poloviční, byl by CR roven 0,5. Další metrikou, kterou budeme používat je BPP, neboli počet bitů na jednotlivý pixel, hodnota BPP se počítá pro komprimovaný obraz podle Rovnice 2.

$$\text{BPP} = \frac{\text{Velikost souboru v bitech}}{\text{Počet pixelů v obrazu}} \tag{2}$$

V neposlední řadě nás bude zajímat i rychlost komprese, kterou můžeme definovat jako počet bytů, či jejích násobků, které dokáže kompresní algoritmus zpracovat za jednotku času. Rychlost komprese si můžeme představit jako mohutnost průtoku dat kompresním algoritmem. V této práci budeme používat jednotky MB/s, kde 1 MB = 1 000 000 B.

Univerzálními kompresními algoritmy, které zde uvedeme, rozumíme algoritmy, které se používají pro kompresi dat všeho druhu. Tyto algoritmy provádějí bezztrátovou kompresi, tedy nedojde ke ztrátě žádné informace. Zvolené metody jsou navrženy tak, aby byly nezávislé na procesoru, operačním systému, či souborovém systému. I z tohoto důvodu jsou používány každodenně v programech jako je například 7-zip.

Námi vybrané metody jsou:

1. gzip
2. lzma
3. bzip2

Během komprese jsou originální soubory převáděny do speciálních formátů, které jsou většinou specifické pro danou metodu. Jeden z jednodušších formátu je například gzip [6], který je využíván stejnojmenným nástrojem. Tato metoda komprese může být také využívaná v HTTP komunikaci. Gzip je založen na metodě DEFLATE [7], která kombinuje dva různé algoritmy, LZ77 [8] a Huffmanovo kódování. Více o Huffmanově kódování bude uvedeno v kapitole 3.4.

Další zvolenou metodou komprese je algoritmus LZMA [9], který znovu kombinuje více algoritmů. Součástí LZMA najdeme algoritmy LZ77, Markův model predikce a aritmetické kódování, přesněji *Range Coding*.

Třetí metodou je algoritmus bzip2 [10], který podobně jako LZMA kombinuje více metod komprese a současně přidává několik transformací. Pomocí těchto transformací se snaží vylepšit výslednou kompresi. Detailní vysvětlení uvedených algoritmů je nad rámec této práce, zde zkoumáme pouze jejich výsledky. Čtenáři, doporučujeme následovat zdroje uvedené v literatuře.

2.2.1 Algoritmus B^{3}D

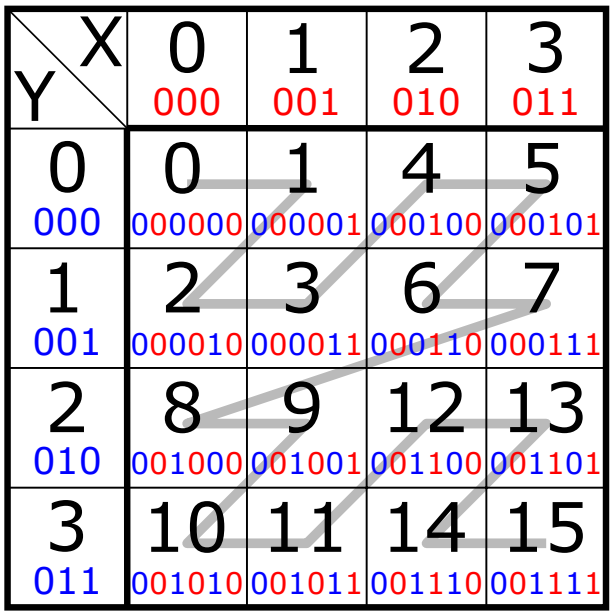
B^{3}D [5] je open source knihovna, která se snaží řešit problém komprese obrazů z mikroskopu. Je napsána v jazyce C++ a využívá technologie CUDA [11]. Tato knihovna nabízí jak ztrátovou, tak i bezztrátovou kompresi, kde ztrátová komprese je řízena velikostí kvantizačního kroku. Velkou výhodou tohoto algoritmu je využití CUDA architektury, kde komprese i dekomprese jsou prováděny na grafických kartách. Toto obecně znamená mnohem větší propustnost.

Bezztrátová varianta se skládá ze dvou částí. V první části se provádí predikce hodnoty pixelu vzhledem k sousedním pixelům. Podobné techniky je využíváno v JPEG kompresi. Následně v části druhé jsou chyby predikce zakódovány pomocí Run-Length kódování a Huffmanova kódování. Ztrátová varianta funguje pomocí využití kvantizačního kroku ve Fourierově prostoru, do kterého je obraz převeden. Tedy hodnoty ve Fourierově prostoru jsou redukovány na určitý počet kvantizačních hodnot. Více o kvantizaci bude uvedeno v následujících sekcích.

2.2.2 Mortonovo kódování

Mortonovo kódování [12] je způsob zobrazení multidimenzionálních dat na data jednodimenzionální. Jedná se o přeuspořádání dat podle Z křivky [12], která jimi prochází. Z křivka spadá do skupiny tzv. prostor vyplňujících křivek. Tyto křivky obecně prochází každým bodem prostoru v n-dimenzionální hyperkrychli. Na Obrázků 2 můžeme vidět jak probíhá zobrazení 2D prostoru na 1D prostor.

Pro každý bod roviny je vypočtena jedna Z souřadnice, pomocí které jsou seřazeny jednotlivé body prostoru. Z souřadnice je vypočtena prolnutím bitů souřadnic X a Y. V našem případě pomocí Z souřadnice přeuspořádáme data obrazu. Podobný způsob přeuspořádání dat se používá pro ukládání textůr na grafických kartách, kde toto indexování vede k optimalizaci přístupů do paměti. Chtěli bychom tedy zjistit, zda by přeuspořádaná data vedle k lepší kompresi, například díky lepší predikci následujících hodnot.



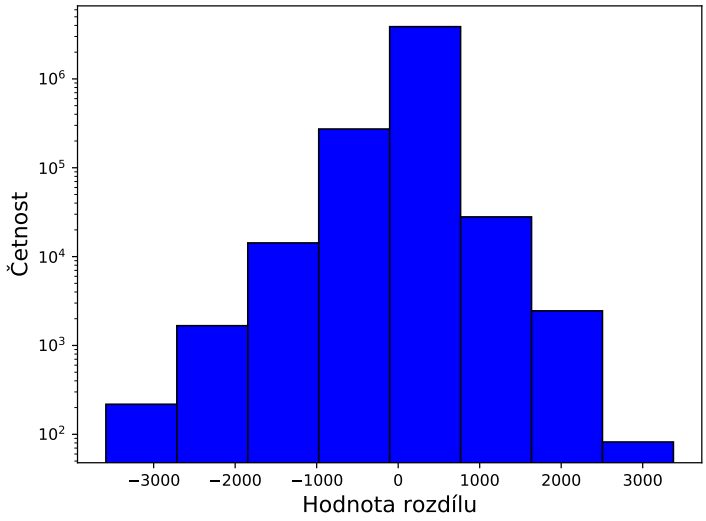
Obrázek 2: Mortonovo kódování pomocí Z křivky

2.2.3 Komprese pomocí rozdílu snímků

Jak již bylo zmíněno dříve, v bioinformatických datasetech se nachází více snímků. Tyto snímky jsou si většinou podobné, a to už z důvodu jejich získání. Většinou se bude jednat o různé řezy stejného biologického vzorku. Můžeme tedy očekávat, že pokud řezy patří do stejného kanálu, tak určitá část obrazu zůstane mezi dvěma sousedními řezy stejná. Řezy musí patřit do stejného kanálu, neboť různé kanály mohou obsahovat zcela rozdílné obrazy, jako jsme viděli na Obrázku 1. Označme si právě kódovaný řez jako I_i, a řez jemu předcházející jako I_{i-1}, rozdíl pixelů, neboli ΔI je definován následovně:

$$\Delta I = I_i - I_{i-1}$$

K získání řezu I_i by nám tedy stačil pouze předcházející řez I_{i-1} a rozdíl ΔI. Nejčastějším typem pixelu v obrazech z mikroskopu je 16bitový pixel, což znamená, že pro uložení rozdílu bychom potřebovali 32 bitů na jeden pixel. Toto by samo o sobě vedlo ke zdvojnásobení velikosti jednoho řezu. Podíváme-li se avšak na distribuci hodnot rozdílu obrazů na Obrázku 3, všimneme si, že většina hodnot leží v oblasti blízko nuly.



Obrázek 3: Distribuce hodnot rozdílu pixelů

Tato vlastnost nám dovolí použít zobrazení, které převede záporná čísla na kladná a budeme znovu schopní využít pouze 16 nebo méně bitů na pixel. Námi zvolená technika zobrazuje záporná čísla na sudé pozice a kladná čísla na liché pozice. Například sekvenci $\{-3, -2, -1, 0, 1, 2, 3\}$ zobrazíme na hodnoty $\{6, 4, 2, 0, 1, 3, 5\}$, dle předpisu uvedeného v Rovnici 3.

$$map(x) = \begin{cases} 0 & \text{pokud} \quad x = 0 \\ 2|x| & \text{pokud} \quad x < 0 \\ 2x - 1 & \text{pokud} \quad x > 0 \end{cases} \tag{3}$$

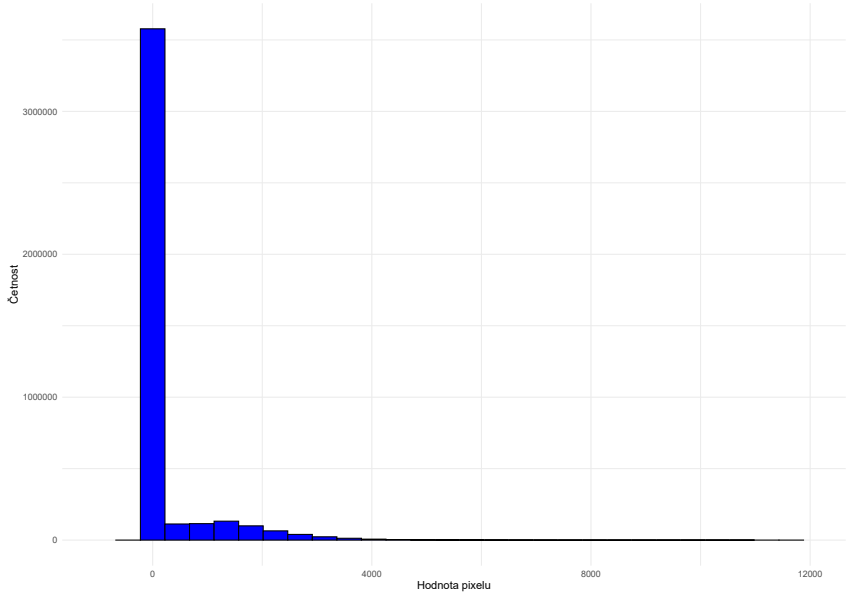
2.3 Popis testovaných dat

Vybrané bezztrátové kompresní algoritmy budeme testovat na obrazech, nacházejících se v CZI souborech. V této sekci si dále popíšeme o jaké obrazy se jedná. V každém CZI souboru se nacházelo několik řezů, ze kterých jsme vždy vybrali jeden na testování. Všechny tyto vybrané snímky mají originálně 16 bitů na pixel. V tabulce 2 se nachází označení snímků, pod kterými je budeme uvádět v následujících obrázcích a grafech.

Označení	Kanál	Z-Index	Rozměr	Typ pixelu
Cells	2	24	1024×800	Gray16
Lz2	1	50	1200×1920	Gray16
Artemia	1	19	2048×2048	Gray16
Llc	1	54	512×512	Gray16

Tabulka 2: Testovací snímky pro bezztrátovou kompresi

Na histogramu v Obrázku 4 můžeme vidět rozložení hodnot pixelů pro snímek Artemia. Všimneme si, že i když může pixel nabývat hodnot od 0 do 65535, tak většina hodnot leží blízko 0. Toto je fenomén, který pozorujeme u všech zkoumaných snímků, a i u dalších, které jsme nevybrali do testování. Je to způsobeno tím, že prostředí, ve kterém se nachází biologický vzorek, tvoří většinovou část obrazu. Toto prostředí je většinou ve výsledném obrázku černé. Obecně jsme pozorovali, že u většiny snímků jsou nejčetnější hodnoty na dolním konci možného rozsahu. Hodnoty uprostřed jsou pak méně časté.



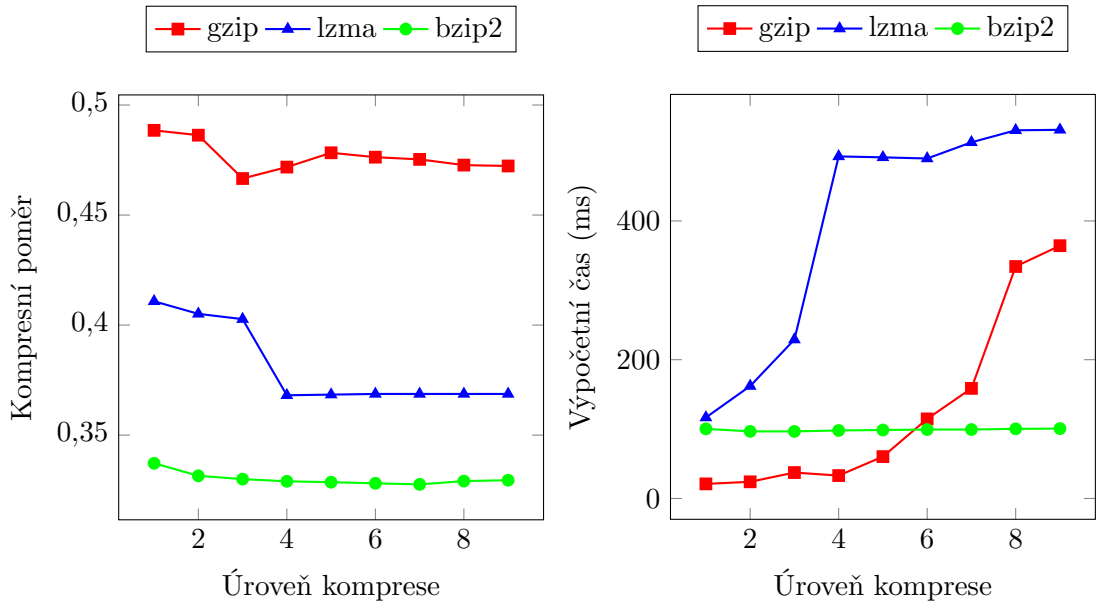
Obrázek 4: Histogram hodnot pixelů CZI řezu (Artemia)

2.4 Analýza výsledků bezztrátových kompresních algoritmů

V této sekci uvedeme výsledky kompresí pro čtyři námi vybrané algoritmy. Budeme zde hodnotit hlavně kompresní poměr (rovnice 1), BPP (rovnice 2) a propustnost algoritmů. U třech námi vybraných algoritmů se dá nastavit tzv. úroveň komprese. Obecně platí, že s vyšší úrovní komprese bychom měli dosáhnout lepších kompresních poměrů. Vzrůstající úroveň, avšak také znamená větší časovou a paměťovou náročnost. Ne vždy je tedy nejlepším řešením použít maximální úroveň. V následujícím grafu na Obrázku 5a si tedy srovnáme kompresní poměry vzhledem k úrovni komprese, test byl proveden na snímku Cells.

Zjistili jsme, že úroveň komprese hraje největší roli pro algoritmus lzma, který vydává lepší výsledky až od 4. úrovně. U algoritmu bzip2 nepozorujeme zlepšení kompresního poměru se vzrůstající úrovní komprese. Avšak hned na první úrovni dosahuje nejlepších výsledků ze tří. V následujících testech budeme používat kompresní úroveň 6, neboť po této úrovni již nedocházelo k žádným znatelným zlepšením. Na následujícím grafu v Obrázku 5b si ještě ukážeme závislost výpočetního času vzhledem k úrovni komprese. U algoritmů gzip a lzma potvrzujeme

naše původní sdělení, že čas komprese roste s úrovní komprese. Dále zjišťujeme, že čas algoritmu lzma je pětinásobně delší než čas algoritmu bzip2. Algoritmus bzip2 pracuje v poměrně konstantním čase, což je velkou výhodou a ideálním případem. Tyto časy komprese jsou průměrem přes několik spuštění kvůli eliminace vnějších vlivů na test.



(a) Kompresní poměr vzhledem k úrovni

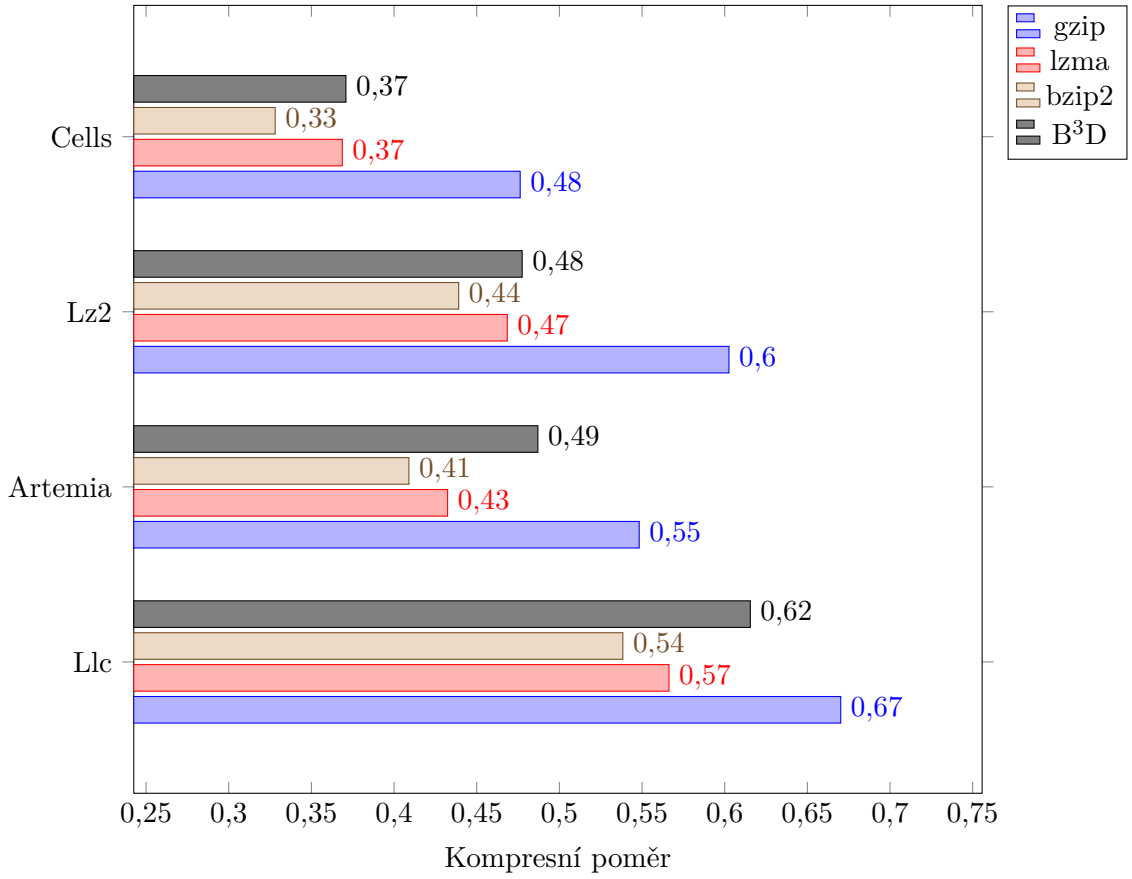
(b) Ča komprese vzhledem k úrovni

Obrázek 5: Vliv úrovně komprese na kompresní poměr a rychlost, snímek Cells

Dále si již otestujeme všechny 4 algoritmy podle dosáhnutého kompresního poměru, který by měl být co nejmenší. Výsledky pro všechny 4 testovací snímky nalezneme v grafu na Obrázku 6. Tento graf znovu potvrzuje, že algoritmus bzip2 vydává nejlepší kompresní výsledky. Na všech snímcích dosáhnul bzip2 nejlepšího výsledku z testovaných algoritmů. Další v pořadí jsou algoritmy B^3D a lzma, které dosahují podobných výsledků. Jako nejhorší hodnotíme nejjednodušší algoritmus gzip. Tyto výsledky jsme očekávali.

Dosud jsme hodnotili samotný výsledek komprese, pokud avšak začneme brát v potaz propustnost v MB/s, tak se nám hodnocení algoritmů změní. Souhrn výsledků testování společně s propustností nalezneme v Tabulce 3. Co se týče propustnosti, tak ta je průměrem přes několik spuštění. Do výpočetního času počítáme pouze samostatnou kompresi dat v paměti, u algoritmu B^3D je navíc přidán čas kopírovaní do paměti grafické karty. Testování probíhalo na počítači s procesorem Intel Core i7-8750H a grafickou kartou NVIDIA GeForce 1060 Max-Q.

Dle této Tabulky 3, hodnotíme algoritmus B^3D jako nejvhodnější. V rámci nejmenšího kompresního poměru v co nejkratším čase dosahuje nejlepších výsledků. Důvod proč vůbec brát propustnost v potaz je velké množství dat. Určitě nechceme, abychom prováděli kompresi delší dobu, než by trvalo přenést ušetřené byty přes internetovou síť. Pokud bychom nebrali rychlost v potaz, tak je algoritmus bzip2 jasným vítězem, neboť konstantně dosahuje nejlepších výsledků



Obrázek 6: Srovnání bezztrátových kompresí mezi více snímky, úroveň komprese 6

a v nejlepším případě zabírá pixel po kompresi pouze průměrně 5,2504 bitů místo původních 16 bitů. V budoucí implementaci by se propustnost dala zlepšit pomocí paralelizace procesu komprese. Více snímků by bylo komprimováno zároveň na různých jádrech procesoru nebo na více grafických kartách.

Dále se podíváme, jak ovlivnilo přeuspořádání dat podle Z křivky výsledné kompresní poměry. Rozdíly kompresního poměru od původního kompresního poměru nalezneme v Tabulce 4, kde kladná hodnota rozdílu kompresního poměru značí zlepšení výsledku daného algoritmu. U prvních dvou algoritmů gzip a lzma pozorujeme lehké zlepšení v rámci setin kompresního poměru. Dále pro algoritmus bzip2 nemělo přeuspořádání žádný smysl. Výsledky jsou spíše horší a algoritmus B^{3}D skončil ve všech případech s horším výsledkem. Zjistili jsme tedy, že toto přeuspořádání nevede vždy k lepším výsledkům, a když už dojde ke zlepšení, tak se nejedná o znatelný rozdíl oproti původním hodnotám. Navíc tato transformace dat sebou přináší další výpočetní nároky.

Jako další metodu, kterou jsme vyzkoušeli je komprese rozdílu řezu ΔI od řezu předcházejícího. V rámci zobrazení hodnot rozdílu jsme také vyzkoušeli ukládat hodnoty do minimálního potřebného počtu bitů, např. 11 bitů. Toto se zdálo jako dobrý nápad, neboť v některých testech

Metoda	Soubor	Kompresní poměr	Bity na pixel	Propustnost (MB/s)
gzip	Cells	0,4763	7,6205	14,2884
	Lz2	0,6026	9,6423	16,4376
	Artemia	0,5483	8,7733	14,7427
	Llc	0,6702	10,7235	15,5729
lzma	Cells	0,3687	5,8984	3,3437
	Lz2	0,4685	7,4960	3,0190
	Artemia	0,4324	6,9189	2,4273
	Llc	0,5663	9,0598	4,9932
bzip2	Cells	0,3281	5,2504	16,4940
	Lz2	0,4391	7,0260	16,1874
	Artemia	0,4090	6,5446	15,6116
	Llc	0,5384	8,6141	14,6997
B^3D	Cells	0,3709	5,9346	898,7660
	Lz2	0,4775	7,6394	279,8587
	Artemia	0,4870	7,7917	614,2986
	Llc	0,6155	9,8472	282,5594

Tabulka 3: Shrnutí bezztrátových kompresí

jsme byli schopni ušetřit až 7 bitů na pixel. Jenže komprese pomoci námi vybraných algoritmů dopadla hůře. Tyto algoritmy si nebyly schopny poradit s bytovou sekvenci, kterou jsme jim předali. V této sekvenci jeden byte obvykle obsahoval informace o dvou různých hodnotách. Toto je pravda, pokud není použitý počet bitů celočíselně dělitelný 8. Zůstali jsme tedy u zobrazení na 16 bitů.

Při testování jsme zjistili, že se tento přístup nevyplácí, neboť výsledky komprese rozdílů jsou horší než výsledky komprese samostatných snímků. Navíc daná metoda sebou přináší větší časové nároky a není univerzální. Testování snímku Lz2 selhalo, neboť pro zakódování rozdílu by bylo třeba více než 16 bitů. Výsledky této komprese spolu s rozdílem kompresního poměru nalezneme v Tabulce 5. Všimneme si, že většina rozdílu je záporných, tedy došlo ke zhoršení.

2.5 Ztrátová komprese pomocí algoritmu B^3D

V poslední sekci první části si uvedeme výsledky ztrátové komprese pomocí algoritmu B^3D. Nejdříve avšak uvedeme metriku, která se standardně využívá v analýze obrazu, bavíme-li se o rozdílu mezi dvěma obrazy, nebo chybě vzniklé při kompresi. Jedná se o průměrnou kvadratickou chybu pixelů neboli MSE. Jestliže máme referenční obraz I s výškou H a šířkou W, kde $I(x, y)$ je hodnota pixelu v bodě (x, y) a dále obraz I_c, který odpovídá referenčnímu obrazu po kompresi, pak se MSE vypočte podle Rovnice 4.

$$\mathrm{MSE} = \frac{1}{WH} \left\{ \sum_{x=0}^{W} \sum_{y=0}^{H} \left[I\left(x, y\right) - I_c\left(x, y\right) \right]^2 \right\} \tag{4}$$

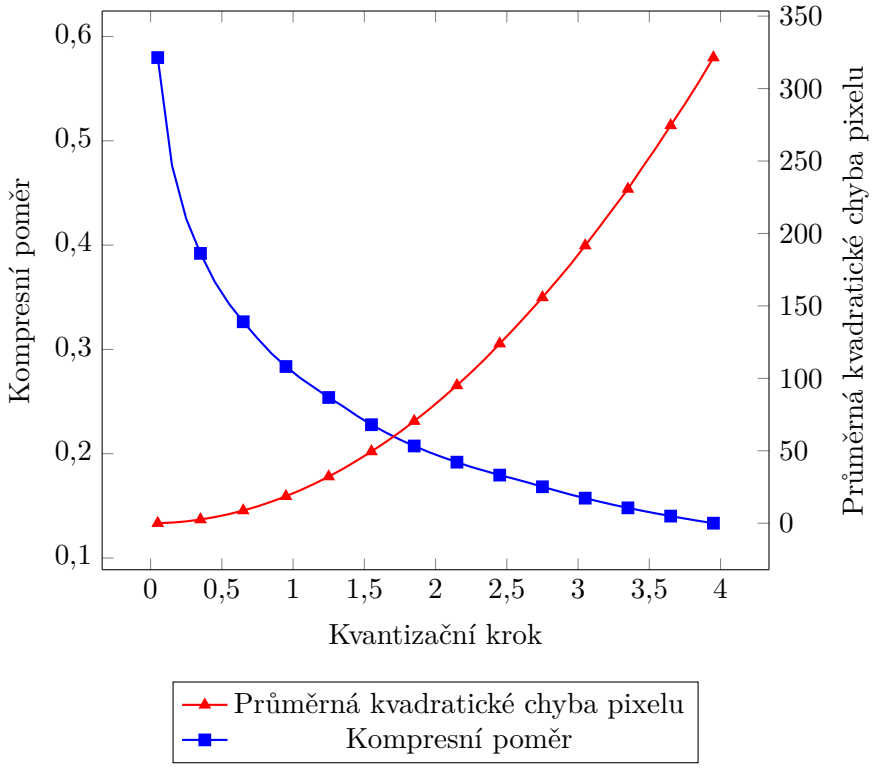
Metoda	Soubor	Kompresní poměr	Rozdíl kompresního poměru
gzip	Cells	0,4606	0,0157
	Lz2	0,5572	0,0454
	Artemia	0,5003	0,0480
	Llc	0,6656	0,0046
lzma	Cells	0,3586	0,0101
	Lz2	0,4463	0,0222
	Artemia	0,4102	0,0222
	Llc	0,5599	0,0064
bzip2	Cells	0,3393	-0,0112
	Lz2	0,4407	-0,0016
	Artemia	0,4062	0,0028
	Llc	0,5406	-0,0022
B^3D	Cells	0,4637	-0,0928
	Lz2	0,5471	-0,0696
	Artemia	0,5197	-0,0327
	Llc	0,6627	-0,0472

Tabulka 4: Kompresní poměry přeuspořádaných dat

Metoda	Soubor	Kompresní poměr I	Kompresní poměr ΔI	Rozdíl
gzip	Cells	0,4763	0,4269	0.0494
	Artemia	0,5483	0,5695	-0.0212
	Llc	0,6702	0,8262	-0.1560
lzma	Cells	0,3687	0,3396	0.0291
	Artemia	0,4324	0,4583	-0.0259
	Llc	0,5663	0,7557	-0.1894
bzip2	Cells	0,3281	0,3359	-0.0078
	Artemia	0,4090	0,4517	-0.0427
	Llc	0,5384	0,7223	-0.1839

Tabulka 5: Komprese ΔI

V našem testování jsme zjistili, že vzniklá chyba se nedá příliš dobře řídit zvoleným kvantizačním krokem ve Fourierově prostoru. Pro každý obraz byla vzniklá chyba, pro daný kvantizační krok jiná. Pro praktické využití by bylo třeba zvolit maximální dovolenou chybu, pro kterou bychom následně museli najít kvantizační krok, což je také přístup zvolený v původním článku tohoto algoritmu [5]. V grafu na Obrázku 7 můžeme vidět, jak klesá kompresní poměr vzhledem ke zvětšujícímu se kvantizačnímu kroku. Zároveň můžeme pozorovat, jak roste MSE. Kvantizační krok nad 4 jsme již nezkoušeli, neboť vzniklá chyba byla již příliš veliká. Vidíme, že pomocí ztrátové komprese můžeme dosáhnout dvojnásobně menších kompresních poměrů s poměrně přijatelnou chybou. Výpočetní čas ztrátové komprese se nijak zvláště neliší od výpočetního času bezztrátové komprese, tedy pořád se jedná o nejrychlejší algoritmus.



Obrázek 7: Ztrátová komprese B^{3}D, snímek Artemia

3 Ztrátová komprese obrazu

Zde se již dostáváme do druhé, hlavní částí práce, ve které se budeme zabývat ztrátovou kompresí obrazu. Hlavně se bude jednat o kvantizační metody. Nejdříve zmíníme standardní metodu komprese obrazu, která se také využívá pro snímky z mikroskopu a bude se jednat o metodu definovanou skupinou JPEG (JPEG komprese). Dále již přejdeme k vysvětlení námi vybraných algoritmů. Zde nejdříve uvedeme jednodušší skalární kvantizace, na kterou následně navážeme vektorovou kvantizací. Výsledky obou těchto metod budou detailně probrány v následující sekci, kde nalezneme výsledky na reálných datech, které budou dopodrobna analyzovány.

3.1 JPEG komprese

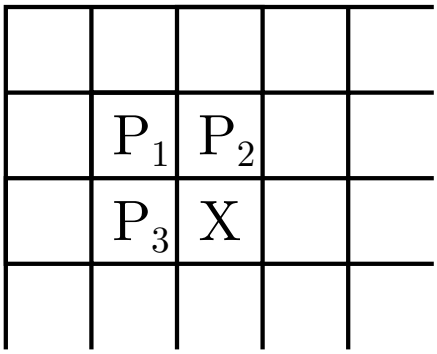
Samotný název JPEG je zkratka skupiny *Joint Photographic Expert Group*, která vytvořila specifikaci komprese nepohyblivých obrazů. Tato specifikace byla v roce 1992 přijata jako standard. Nároky uvedené ve standardu byly následující: kompresní poměr na úrovni nejlepších kompresních metod, kompresi musí být možno použít na jakýkoliv typ obrazu, výpočetní složitost komprese by měla být co nejnižší a měly by být k dispozici různé módy komprese. Těmito módy byly:

- Sekvenční kódování – všechny komponenty obrazu jsou zakódovány jedním průchodem.
- Progresivní kódování – kódování několika průchody, kde postupně roste kvalita.
- Bezztrátové kódování – kódování bez ztráty obrazové informace.
- Hierarchické kódování – obraz je zakódován v několika rozlišeních. Menší rozlišení je dostupné dříve, než dojde k dekompresi vysokého rozlišení.

Bezztrátová metoda JPEG komprese je založena jednoduché predikci hodnoty pixelu. Hodnota pixelu X je predikována na základě maximálně tří sousedních pixelů P_1, P_2, P_3. Tyto pixely můžeme vidět na Obrázku 8a. Možné předpisy prediktorů jsou pak uvedeny v Tabulce 8b. Predikovaná hodnota je následně odečtena od skutečné hodnoty a rozdíl je zakódován pomocí entropického kódování. Používá se buď Huffmanovo nebo Aritmetické kódování [13]. Toto je stejná technika, která byla použitá v algoritmu B^{3}D, sekce 2.2.1.

Ztrátová JPEG komprese je už složitější. Postupně uvedeme kroky, které jsou nad vstupním obrazem provedeny. Většina obrazů je uložena v barevném prostoru *RGB*, který není vhodný pro kompresi. Lidské oko je citlivější na změny jasu než na změnu barvy. Proto je obraz v prvním kroku převeden z barevného prostoru *RGB* do jednoho z prostorů *YUV*, *YIQ* či YC_bC_r. V těchto prostorech nese složka Y jas a zbylé dvě složky jsou barvonosné.

Dále se často provádí redukce objemu barvonosné informace, tj. provede se snížení počtů vzorků v barvonosných rovinách. Typicky je počet barvonosných vzorků redukován dvojnásobně v obou dimenzích. Což znamená, že pro každý pixel máme informaci ohledně Y složky, ale zbylé



(a) Sousední pixely v bezztrátové JPEG kompresi

Prediktor	Předpis
0	X
1	P_3
2	P_2
3	P_1
4	$P_3 + P_2 - P_1$
5	$P_3 + ((P_2 - P_1)/2)$
6	$P_2 + ((P_3 - P_1)/2)$
7	$(P_3 + P_2/2)$

(b) Možné předpisy predikce hodnoty pixelu

Obrázek 8: Bezztrátová JPEG komprese

dvě barvonosné složky jsou vzorkovány po 2×2 blocích (např. pro 4 Y hodnoty máme 1 C_b a 1 C_r hodnotu). Typicky volba kvality JPEG souboru řídí tuto redukci.

Následně jsou všechny roviny rozděleny do bloků velikosti 8×8 a jsou zpracovávány zvlášť, ale stejným způsobem. Ještě před aplikováním DCT (diskrétní kosinová transformace), jsou hodnoty v blocích posunuty tak, ať je střední hodnota rovna nule. Toto se provádí, neboť výsledek DCT (kosinová křivka), je také definován kolem nuly, v intervalu od -1 do 1. Bavíme-li se o 8 bitových pixelech, tak jsou posunuty do intervalu $\langle -128; 127 \rangle$. Na tyto upravené bloky je použita dopředná DCT. Více informací o téhle operaci může čtenář najít v [14], ze kterého taky vycházíme při popisu ztrátové JPEG komprese.

Po provedení DCT jsou bloky kvantovány pomocí kvantizační tabulky o rozměrech 8×8. Kvantování dělí hodnotu v bloku hodnotou v kvantizační tabulce. Kvantizace je založena na pozorování, že některé frekvence, zejména vysoké, lze reprezentovat dosti hrubě [14]. Pro jasovou a barvonosnou rovinu jsou použity různé kvantizační tabulky, příklad tabulky pro kvantování barvonosné roviny je na Obrázku 9a.

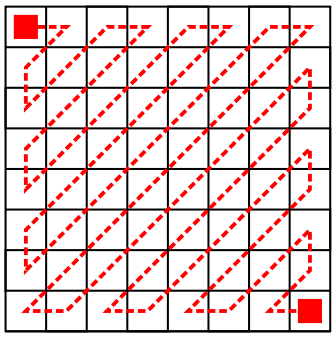
Hodnoty po kvantování jsou následně převedeny do intermediální sekvence pomocí ZIG-ZAG průchodu, který je naznačen na Obrázku 9b. Nenulové hodnoty jsou kódovány pomocí trojice (R, S, A), kde R je počet nul před nenulovou hodnotou, S je počet bitů A hodnoty a A je samotná hodnota amplitudy.

Intermediální sekvence je v poslední fázi zakódována pomocí Huffmanova kódování. Kvantizační tabulka, je spolu s informaci potřebnou pro rekonstrukci Huffmanova stromu, uložena v komprimovaném souboru.

Postupem času vznikaly další verze této metody, jako je například JPEG2000 nebo JPEG XR, který se může používat například v CZI souborech. JPEG XR jako standard přinesl podporu pro více různých typů pixelů, jako jsou například 16bitové pixely, nebo pixely, kde je hodnota číslem s plovoucí desetinnou čárkou. Dále je v tomto novějším standardu zabudovaná podpora pro HDR obrazy a slibuje lepší kompresní výsledky [15].

$$\begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

(a) Kvantizační tabulka pro barvonosnou rovinu



(b) ZIG-ZAG průchod

Obrázek 9: JPEG komprese

3.2 Skalární kvantizace

Kvantizace je jedna z nejjednodušších a nejobecnější metod ztrátové komprese. Je to proces, při kterém jsou hodnoty vstupní veličiny, ať už diskrétní nebo spojité, omezeny na určitý počet diskrétních hodnot. Kvantizace nalézá v oboru komprese dat dvou různých využití. Prvním využitím je převod velkých číselných hodnot na hodnoty menší. Tyto menší hodnoty zabírají méně místa (bitů), a proto tato operace vede ke kompresi. Zároveň v malých hodnotách je obsaženo méně informace, a proto je tato komprese ztrátová. Tohoto prvního pozorování využijeme při kompresi obrazu. Druhým využitím kvantizace je převod analogových dat na data digitální s malými hodnotami [16]. Pří skalární kvantizaci jsou převáděny postupně jednotlivé skaláry, např. vztah mezi sousedními pixely není nijak brán v potaz.

Tak jako u všech kompresních metod, tak i kvantizace má svůj kompresní a dekompresní algoritmus. Kvantizace zobrazuje jednotlivé hodnoty vstupního signálu na omezený počet kvantizačních hodnot. Originální hodnota je zobrazována na slovníkovou hodnotu podle kvantizačního intervalu, do kterého spadá. Tato operace je ireverzibilní. Kvantizační hodnoty se dají nazvat slovníkovými, neboť jich je omezený počet a ve finále jsou uloženy ve slovníku. Samotná operace zobrazení je velice jednoduchá. Problém leží v nalezení co možná nejlepšího slovníku. Špatně nalezené slovníkové hodnoty budou vést ke vzniku velké kompresní chyby. Kompresní chybu, nebo také kvantizační šum, budeme měřit podle MSE (Rovnice 4). Nalezeným slovníkovým hodnotám jsou přiřazeny co nejmenší indexy, které kompresní algoritmus zapisuje do proudu komprimovaných dat. Dekomprese je už jednoduchá, neboť se jedná pouze o vyhledání hodnoty ve slovníku, podle jejího indexu. Počet bitů potřebný pro zapsání jednoho indexu se řídí velikostí slovníku. Jestliže je velikosti slovníku L, pak je na jeden index třeba $\lceil \log_2 L \rceil$ bitů.

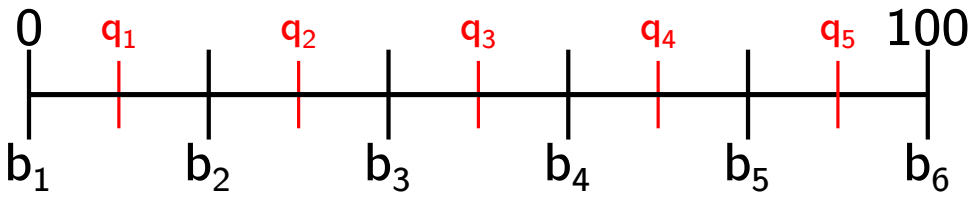
Dále si zde uvedeme další metriku, pomocí které se dá měřit kvalita ztrátové komprese signálu. Touto metrikou je PSNR. PSNR označuje špičkový poměr signálu k šumu a vyjadřuje poměr mezi maximální hodnotou signálu a energii šumu. Šumem rozumíme chybu, která vznikla ztrátovou kompresí. Veličina této metriky je udávána v decibelech (dB) a k jejímu výpočtu po-

třebujeme v našem případě znát hodnotu MSE. Jestliže $MAX(I)$ je maximální hodnota signálu, tak se PSNR vypočte podle Rovnice 5. $MAX(I)$ je pro 16 bitový pixel rovno 65535. Cílem je hodnotu PSNR maximalizovat.

$$\text{PSNR} = 10 \cdot \log_{10} \left[\frac{MAX(I)^2}{\text{MSE}} \right] \tag{5}$$

Jak jsme již zmínili, vstupní hodnota signálu je zobrazována podle intervalu, do kterého spadá. Tyto intervaly jsou ohraničeny okrajovými body, které budeme značit jako b_i. Samotné kvantizační hodnoty budeme značit pomocí q_i. Nejjednodušším typem kvantizátoru je uniformní kvantizátor, pro který platí, že jsou všechny intervaly stejně velké. Na Obrázku 10 můžeme vidět příklad okrajových a kvantizačních bodů, pro uniformní kvantizaci hodnot z intervalu 0 až 100 na 5 diskrétních hodnot. Hodnota x je mapována podle předpisu:

$$q_i \iff x \geq b_i \wedge x < b_{i+1} | i \in \{1, 2, 3, 4, 5\}$$



Obrázek 10: Uniformní skalární kvantizace

Kvantizátory s uniformními intervaly se hodí pouze pro data, kde všechny vstupní hodnoty mají stejnou pravděpodobnost výskytu. Pro ostatní případy, kdy data neodpovídají rovnoměrnému rozdělení, bude vzniklá kvantizační chyba velká. Je tedy třeba zvolit intervaly s různými velikostmi. Kvantizátorům, které mají intervaly s různými velikostmi se říká neuniformní neboli nerovnoměrné.

Cílem neuniformních kvantizátorů je se co nejvíce přiblížit rozdělením kvantizačních hodnot k rozdělení vstupních dat. Intervaly by měly být užší v oblastech, kde je větší pravděpodobnost výskytu hodnot, a naopak širší v oblastech s malou pravděpodobnosti výskytu. Například v obrazu, kde je většina hodnot blízko nuly, chceme mít co nejvíce kvantizačních hodnot blízko nuly. Toto je provedeno za cenu větší chyby při kvantizaci velkých hodnot.

Úroveň komprese je při skalární kvantizaci daná velikosti slovníku L. S více hodnotami jsme schopni lépe zrekonstruovat obraz, za cenu větší velikosti komprimovaného souboru. Velikost slovníku jsme se v našem případě rozhodli určovat podle počtů bitů, pomocí kterých je se dá adresovat jedna slovníková hodnota.

3.2.1 Lloyd–Max algoritmus

V ideálním případě bychom při tvorbě slovníku znali pravděpodobnostní model vstupních dat. Tento model, avšak skoro nikdy při kompresi obrazu neznáme a musíme využít algoritmu, který je schopen slovník vytvořit. Přímý výpočet kvantizačních hodnot a okrajových bodů se dá popsat následujícími dvěma rovnicemi:

$$q_i = \frac{\int_{b_i}^{b_{i+1}} x f_X(x) dx}{\int_{b_i}^{b_{i+1}} f_X(x) dx} \qquad i = 0 \ldots (N-1) \tag{6}$$

$$b_j = \frac{q_j + q_{j-1}}{2} \qquad j = 1 \ldots (N-1) \tag{7}$$

V těchto rovnicích je kvantizační hodnota geometrickým středem pravděpodobnostní funkce v daném intervalu. Okrajové body intervalů jsou pak umístěny mezi tyto geometrické středy. Bohužel jsou tyto rovnice na sobě závislé. Pro vyřešení Rovnice 6, potřebujeme znát výsledek Rovnice 7 a naopak. Iterativní řešení těchto rovnic je známo jako Lloyd–Max algoritmus [17]. V tomto algoritmu jsou integrály nahrazeny sumou a hustota pravděpodobnosti je nahrazena absolutní četnosti výskytu dané hodnoty. Kvantizační hodnoty jsou postupně s každou iterací posouvány do míst s větší pravděpodobností výskytu vstupních dat. Jestliže máme N kvantizačních hodnot q a $N + 1$ okrajových bodů, pak se dá základní algoritmus shrnout v těchto krocích:

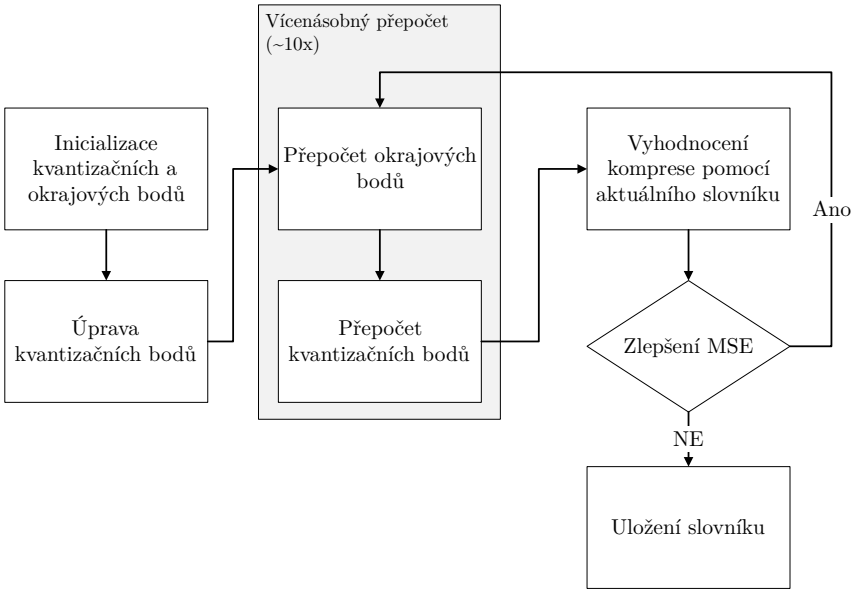
1. vygenerují se počáteční kvantizační hodnoty q_i, pro $i = 0 \ldots N - 1$,
2. vypočítají se okrajové hodnoty b_j, podle Rovnice 7,
3. dle okrajových hodnot b_j se přepočítají q_i, podle Rovnice 6,
4. opakuj 2. a 3. krok dokud dochází ke snížení kvantizační chyby.

Algoritmus Lloyd–Max potřebuje pro vytvoření ideálního slovníku trénovací data. Tyto trénovací data musí odpovídat nebo být alespoň podobná vstupnímu obrazu. My jsme implementovali algoritmus pro kompresi 16 bitových snímků, což znamená, že vstupní data leží v intervalu $\langle 0; 65535 \rangle$. Avšak celý tento interval se nevyplatí vždy využít, místo toho hledáme ideální slovník v intervalu $\langle I_{min}; I_{max} \rangle$, kde I_{min}, resp. I_{max} je nejmenší nalezená hodnota pixelu, resp. největší nalezená hodnota pixelu. Díky této úpravě nebudeme vytvářet kvantizační hodnoty, které by nebyly nikdy využity.

Počáteční kvantizační hodnoty q jsou rovnoměrně vygenerovány na prohledávaném intervalu. Následně jsou nastaveny okrajové body, kde první okrajový bod b_0 je roven minimální nalezené hodnotě pixelu a poslední okrajový bod je roven maximální nalezené hodnotě pixelu. Dále jsou kvantizační hodnoty ještě přesunuty v rámci svého intervalu na pozici s největší absolutní četností. Toto je naše úprava, která není součástí originálního algoritmu, vedoucí k rychlejšímu

nalezení ideálních hodnot. Po tomto kroku již probíhá iterativní vylepšování slovníku, podle 4. kroku algoritmu.

Po každém přepočítání q a b hodnot by mělo dojít k vyhodnocení nalezeného slovníku. Toto vyhodnocení sebou, avšak nese poměrně velké časové nároky pro velkou trénovací sadu. Proto v naší implementaci nejdříve provádíme přepočet ve vnitřním cyklu s n iteracemi. Až po těchto n iteracích dojde k vyhodnocení aktuálního slovníku. Tento vícenásobný přepočet můžeme provést, neboť když se hodnoty q blíží k ideální hodnotě, tak je jejich přesun pomalejší. Nemůže tedy nastat situace, že by hodnoty přeskočili ideální řešení, čímž by došlo ke skokovému zhoršení. Postup celého algoritmu se dá shrnout diagramem na Obrázku 11.



Obrázek 11: Diagram Lloyd–Max algoritmu

3.3 Vektorová kvantizace

Druhou naší metodou, kterou vyzkoušíme na ztrátovou kompresi bioinformatických dat je vektorová kvantizace, která je zobecněním kvantizace skalární. Princip vektorové kvantizace je založen na dvou pozorováních. Zaprvé, komprese řetězců dat, raději než individuálních symbolů, vede k lepším kompresním výsledkům. Více vstupních hodnot je seskupeno do bloků, které jsou následně kódovány. Zadruhé, mezi sousedními hodnotami existuje vztah, hodnoty jsou korelované. Například v obrazu je velká šance, že pixel bude mít stejnou hodnotu jako jeho sousední pixely [16].

Samotný nápad, že komprese sekvence, místo komprese symbolů může vést k lepším výsledkům, byl nejprve představen Claudem Shannonem. Tato myšlenka byla následně potvrzena, když byly postupně kódovány delší a delší sekvence symbolů najednou. Z tohoto vychází, že použitím této metody by mělo dojít k alespoň nějakému zlepšení oproti skalární kvantizaci [17].

Nejjednodušší algoritmus by vytvořil slovník všech možných bloků velikosti N a pří kódování by pouze zapisoval index do tohoto slovníku. Tento přístup je velmi neefektivní, neboť velikost slovníku by byla obrovská. Jestliže jeden prvek vektoru zabírá k bitů, pak by velikost slovníku byla 2^{kN}. Dalším přístupem by bylo přidávat nově objevené vektory z obrazu do slovníku postupně. Problémem tohoto přístupu je postupně rostoucí slovník a s tím i rostoucí velikost indexu v bitech. Z důvodu, že jsou oba tyto přístupy vytvářející slovníky všech možných vektorů vstupního obrazu nepoužitelné, je vektorová kvantizace ztrátová, neboť musíme vytvořit slovník omezené velikosti [16]. Přístup, který bude využívat dále popsaný algoritmus LBG je založeny na analýze trénovacích dat, ve kterých se budeme snažit najít L nejuniverzálnějších vektorů, které nejlépe popisují vstupní obraz.

Po nalezení slovníku je práce algoritmu již přímočará. Pro každý vstupní blok pixelů, musí být nazelen nejpodobnější blok ve slovníku a do komprimovaného proudu dat být zapsán jeho index. Velikost indexu je stejně jako u skalární kvantizace $\lceil \log_2 L \rceil$ bitů. Dekompresní algoritmus následně pouze vyhledá blok ve slovníku podle jeho indexu. Navíc oproti skalární kvantizaci nahrazujeme jeden index N hodnotami, takže je dekomprese ještě rychlejší. Podobnost bloku vstupních dat B a slovníkového bloku C měříme pomocí vzdálenosti mezi nimi. Tato vzdálenost je většinou určena jednou z rovnic uvedených v Rovnici 8.

$$\begin{aligned} d_1(B,C) &= \sum_{i=0}^{N} |b_i - c_i| \\ d_2(B,C) &= \sum_{i=0}^{N} (b_i - c_i)^2 \\ d_3(B,C) &= \mathrm{MAX}_{i=0}^{N} |b_i - c_i| \end{aligned} \tag{8}$$

V celé této sekci se bavíme o vektoru majícím N prvků. Tyto vektory mohou být z klasických dvoudimenzionálních obrazů vytvořeny pomocí dvou různých způsobů. První základní způsob je vzít postupně blok N pixelů v řadě. Druhým způsobem, který lépe využívá výhod dvoudimenzionálních obrazů, je vzít blok rozměrů 3×3 či 5×5, obecně $X \times Y$. Většinou se volí $X = Y$. Pomocí těchto dvoudimenzionálních bloků lépe využíváme vlastnost podobnosti sousedních pixelů. Tato myšlenka se dá také využít dále na volumetrická data, kde bychom byli schopni vytvořit krychle rozměrů $X \times Y \times Z$. Samotný algoritmus LBG je nezávislý na této volbě, neboť bloky jsou vždy převedeny na základní vektory, se kterými algoritmus pracuje.

3.3.1 LBG algoritmus

LBG algoritmus, celým jménem Linde–Buzo–Gray [18], je algoritmus, který se snaží najít co nejlepší slovník dané velikosti, a to pomocí adaptace na vstupní trénovací data. Celý tento algoritmus je založen na shlukovacím algoritmu k-means. Algoritmus k-means dostane na vstupu trénovací vektory, které rozdělí do k shluků. Ve shlucích jsou podobné vektory. V první iteraci je vybráno k náhodných vektorů, reprezentující shluky. Ostatní vektory jsou přiřazeny

k nejbližšímu shluku (měřeno např. pomocí euklidovské vzdálenosti). Následně je v rámci každého shluku vypočten geometrický střed a příslušnost všech vektorů do shluků je přepočítána. Toto se opakuje, dokud se mění prvky ve shlucích. Je to podobný přistup, který byl využit v rámci Lloyd–Max algoritmu v kapitole 3.2.1.

Konkrétní algoritmus LBG si popíšeme pomocí aplikace na kompresi obrazu a budeme vycházet z [16]. Trénovacími vektory pro nás budou bloky velikosti N, které získáme ze vstupního obrazu.

0. Vytvoříme počáteční slovník o velikosti L, obsahující vektory délky N. Dále si zvolíme minimální hodnotu zlepšení ϵ a nastavíme hodnotu $D_0 = \infty$.

1. Trénovací vektory postupně přiřadíme k nejbližšímu slovníkovému vektoru. Vzdálenost je měřena jednou z metrik uvedených v Rovnicích 8. Uskupení slovníkového vektoru a trénovacích vektorů budeme nazývat jako shluk.

2. V rámci každého shluku je vypočtena průměrná vzdálenost trénovacích vektorů od slovníkového vektoru. Tyto vypočtené hodnoty jsou znova zprůměrovány, za vzniku hodnoty D_i, neboli zkreslení aktuálního slovníku. Spodní index i značí iteraci algoritmu.

3. Je provedeno vyhodnocení, zda došlo k vylepšení slovníku podle Rovnice 9. Jestliže tato podmínka není splněna, tak algoritmus končí.

$$\frac{D_{i-1} - D_i}{D_i} \leq \epsilon \tag{9}$$

4. V každém shluku je vypočítán nový geometrický střed, který se stává novým slovníkovým vektorem. Tento geometrický střed je vypočítán jako průměrný vektor ze všech trénovacích vektorů v daném shluku. Po této operaci máme nový vylepšený slovník, který budeme dále zlepšovat. Pokračujeme prvním krokem.

Pro praktické využití algoritmu je ještě třeba vyřešit pár problémů. Prvním problémem je velká závislost kvality finálního slovníku na počátečním slovníku. Tento algoritmus nijak nezaručuje, že nalezne optimální řešení a volba počátečního slovníku je velmi důležitá. V originálním práci [18], byla navrhnuta technika postupného dělení slovníku na větší slovník. V dané době (1980) byla tato metoda označena za velmi výpočetně náročnou, ale aktuálně jí můžeme bez problému využít. Dělící technika funguje následovně:

0. Vytvoříme slovník velikosti 1, kde daný vektor je průměrem všech trénovacích vektorů.

1. Každý slovníkový vektor $\vec{c}$ rozdělíme na dva vektory pomocí perturbačního vektoru $\vec{e}$, podle $\vec{c} \pm \vec{e}$. Po tomto rozdělení budeme mít dvakrát větší slovník. Druhou variantou je přenést všechny staré vektory do nového, většího slovníku. Poté jsou ještě všechny tyto přenesené vektory rozděleny pomocí $\vec{c} + \vec{e}$.

2. Na aktuální slovník použijeme LBG algoritmus, abychom jej co nejvíce zlepšili.

3. Jestliže máme slovník požadované velikosti tak algoritmus končí, jinak se přesuneme na první krok a pokračujeme s dělením.

Tato metoda dělení funguje velmi dobře a již nemusíme řešit ruční volbu počátečního slovníku pro algoritmus LBG. Avšak problémem je správná volba perturbačního vektoru $\vec{e}$. Ručně zvolený $\vec{e}$ by mohl teoreticky vést k dobrému rozdělení v rámci jednoho vektoru, ale ke špatnému u všech ostatních. Proto jsme se rozhodli použít dynamickou volbu, která byla popsána v [19]. Tato technika tvoří unikátní perturbační vektor pro každý shluk zvlášť. Vektor $\vec{e}$ je tvořen podle trénovacích vektorů daného shluku, přesněji podle maximálních a minimálních hodnot v různých dimenzích vektorů. Jestliže je dimenze označována pomocí N, tak je perturbační vektor $\vec{e}$ pro každý shluk vypočten podle Rovnice 10.

$$\vec{e} = \left(\frac{w^1_{\max} - w^1_{\min}}{N}, \frac{w^2_{\max} - w^2_{\min}}{N}, \ldots, \frac{w^N_{\max} - w^N_{\min}}{N} \right) \tag{10}$$

V této rovnici $w^i_{\max}$, resp. $w^i_{\min}$ značí maximální, resp. minimální hodnotu i-té složky trénovacích vektorů daného shluku. Jediným problémem, který může nastat je prázdný shluk, kde žádný trénovací vektor nepřísluší vektoru slovníkovému. Pro tento shluk je $\vec{e}$ vytvořen náhodně.

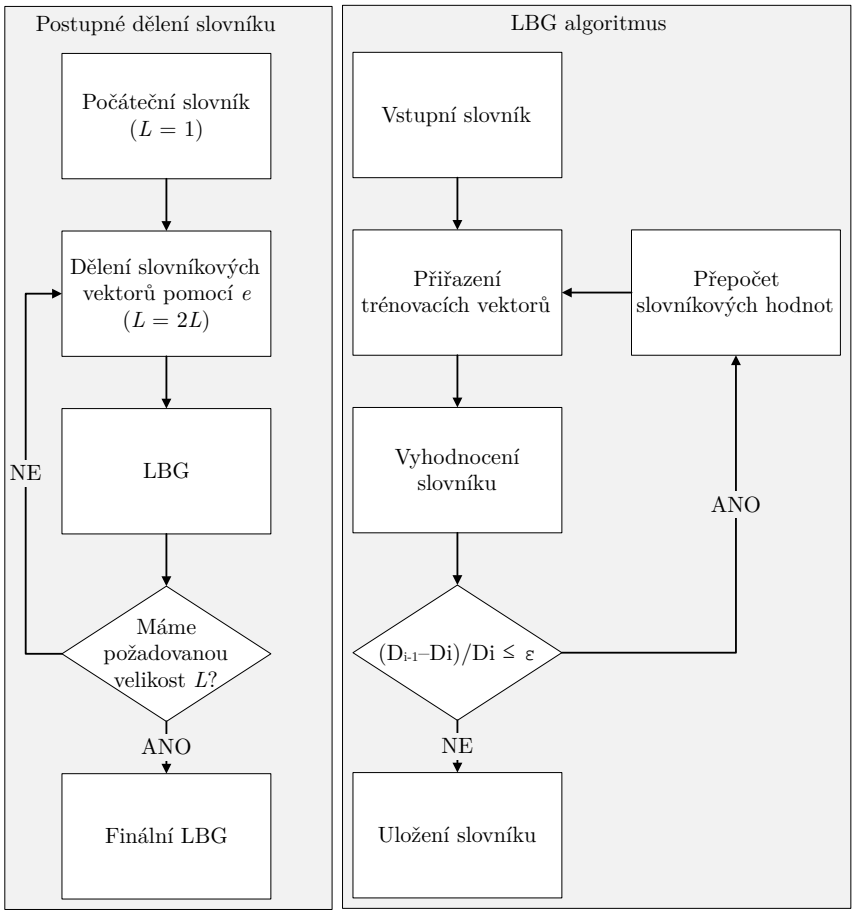
Druhým problémem u samotného LBG algoritmu je vznik tzv. prázdného slovníkového vektoru. Tomuto vektoru nepřísluší žádné trénovací vektory a je tedy ve slovníku navíc. Prázdný shluk zabírá místo pro jiný vektor, který by mohl pomoci vytvořit optimální slovník. Řešením tohoto problému je smazat tento prázdný vektor ze slovníku a nahradit jej. Jako náhradní vektor je zvolen náhodný trénovací vektor z největšího shluku (největší počet trénovacích vektorů). Trénovací data největšího shluku jsou následně rozdělena mezi tyto dva vektory. Na diagramu v Obrázku 12 můžeme vizuálně pozorovat, jak fungují oba algoritmy, LBG i postupné dělení slovníku.

3.4 Huffmanovo kódování

Huffmanovo kódování [20] je velmi využívána metoda komprese dat. Tato metoda vytváří pro abecedu symbolů, známých pravděpodobností, nejlepší bitové kódy proměnné délky. Toto kódování může být v kompresním programu využito samostatně, nebo jak je tomu ve většina případů jako další kompresní krok. Později zmíněného využití použijeme v naší aplikaci.

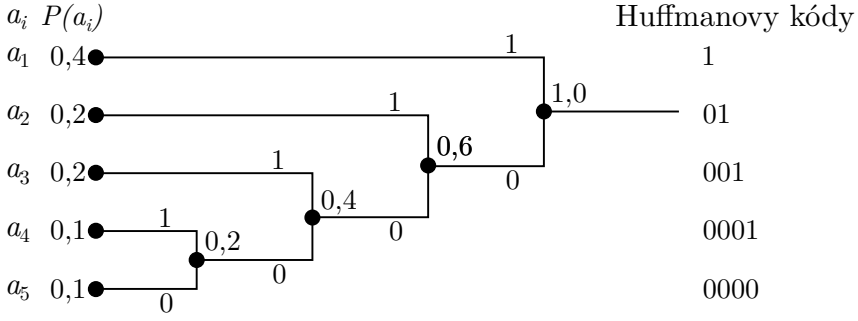
Tento algoritmus přiřazuje nejpravděpodobnějším symbolům nejkratší kódy, a proto je nutné znát pravděpodobnost symbolů abecedy. V ideálním případě by byly pravděpodobnosti symbolů rovny záporným mocninám dvojky. Huffmanovy kódy jsou tvořeny při budování stromové struktury.

Na začátku jsou symboly vstupní abecedy seřazeny sestupně dle jejích pravděpodobností. Ze symbolů jsou následně vytvořeny vrcholy stromu, které jsou vloženy do seznamu. Dále jsou ze seznamu odstraněny dva vrcholy s nejmenší pravděpodobností. Pomocí těchto dvou vrcholů



Obrázek 12: Diagram LBG algoritmu

vytvoříme nový vrchol, potomka, jehož pravděpodobnost se rovná součtu pravděpodobností dvou odstraněných vrcholů. Potomka vložíme do seznamu. Toto odebírání a slučování probíhá, dokud v seznamu nezůstane pouze jediný vrchol, kořen stromu [16]. Poté jsou hrany mezi vrcholy ohodnoceny bity 0 a 1. Nakonec postupně projdeme strom od kořene do všech listů, čímž vytvoříme binární kódy symbolů. Příklad budování Huffmanova stromu pro symboly a_i můžeme vidět na Obrázku 13. Na pravé straně obrázku můžeme vidět vytvořené binární kódy.



Obrázek 13: Huffmanovo kódování

Kompresní algoritmus tedy převádí vstupní symboly na binární kódy, které jsou zapisovaný do komprimovaného proudu dat. Dekompresní algoritmus následně čte z proudu po bitech. Pomocí těchto bitů se posouvá v binárním Huffmanově stromu, dokud nedojde do listu. V listu se nachází samotný symbol. Je tedy zřejmé, že při dekompresi musí být zrekonstruován stejný Huffmanův strom jako používal kompresní algoritmus. Z tohoto důvodu musí být spolu s komprimovanými daty zapsána vedlejší informace, která bude využita pro rekonstrukci Huffmanova stromu. Většinou se jedná buď o četnosti jednotlivých symbolů nebo o samotný strom v binární podobě.

Střední délka Huffmanova kódu leží v intervalu $\langle E; E+1\rangle$, kde E je střední hodnota entropie vstupních symbolů. Střední délka kódu je rovna entropii právě tehdy, když se pravděpodobnosti rovnají záporným mocninám dvojky.

3.5 Vlastní kompresní nástroj

V této sekci uvedeme informace ohledně vlastního kompresního nástroje, který jsme v rámci práce vytvořili. Pod pojmem nástroj rozumíme knihovnu a konzolový program, které jsme napsali v programovacím jazyce Java. Tento jazyk byl zvolen, jelikož je hojně využíván v bioinformatické oblasti, a to například v programech Fiji [2] a BigDataViewer [1]. Do druhého zmiňovaného programu bychom rádi naši knihovnu v budoucnu napojili.

Námi vytvořený nástroj obsahuje jak kompresní, tak i dekompresní algoritmy a zaměřuje se na obrazy s 16 bity na pixel, které jsou nejčastější v bioinformatických aplikacích. V tomto nástroji jsou implementovány metody ztrátové komprese obrazu. Jedná se specificky o dříve zmíněné skalární kvantizaci (sekce 3.2) a vektorovou kvantizaci (sekce 3.3). Náš nástroj dovoluje kompresi obrazových dat, které se nacházejí v několika formátech. Sami jsme implementovali funkce pro čtení surových RAW obrazových dat. Co se týče ostatních formátů, například TIFF, tak pro ně využíváme knihovny SCIFIO [21].

Při kompresi se náš nástroj stará jak o samotnou kompresi, tak o vytvoření souboru ve speciálním formátu, který jsme nazvali QCMP. Tento binární formát je definován svou hlavičkou a její strukturu můžeme vidět v Tabulce 6. V této hlavičce I_X značí velikost obrazu v dimenzi X, neboli šířka obrazu. Podobně I_Y výšku obrazu a I_Z je počet rovin, řezů datasetu. Následující tři hodnoty značí velikosti kvantizačního vektoru při vektorové kvantizaci. Velikosti rovin je pole délky I_Z, ve kterém je uložena informace o počtu bytů, potřebných k zakódování jednotlivých rovin.

Za hlavičkou již následují komprimovaná obrazová data. Obě implementované metody jsou slovníkové, neboť vytvářejí slovník určitých hodnot, nebo vektorů. Obraz je pomocí slovníku komprimován. Jejich výstupem je tedy proud indexů, odkazující na slovníkové záznamy. Pro zlepšení kompresního poměru jsou tyto indexy kódovány pomocí Huffmanových binárních kódů.

V našem programu podporujeme tzv. globální slovník (viz typ slovníku v Tabulce 6). Globální slovník je jeden slovník, který je použit pro kompresi všech rovin datasetu. Pokud není zvolena tato možnost, je pro každou rovinu vytvořen vlastní slovník. Slovník pro každou ro-

Vzdálenost	Velikost	Typ	Možné hodnoty	Poznámka
0	8	ASCII řetězec	*QCMPFILE*	Podpis souboru
8	1	Byte	0,1,2,3	Typ kvantizace
9	1	Byte	1 - 255	BPP
10	1	Byte	0,1	Typ slovníku
11	2	UInt16	0 - 65535	I_X
13	2	UInt16	0 - 65535	I_Y
15	2	UInt16	0 - 65535	I_Z
17	2	UInt16	0 - 65535	V_X
19	2	UInt16	0 - 65535	V_Y
21	2	UInt16	0 - 65535	V_Z
23	$4I_Z$	UInt32[]		Velikosti rovin
$23 + 4I_Z$		Byty		Data rovin a slovníků

Tabulka 6: Hlavička QCMP formátu

vinu znamená větší časovou náročnost, jelikož musí být vždy nalezen. Zároveň znamená také větší komprimovaný soubor, neboť všechny slovníky musí být zapsány. Naproti tomu při použití globálního slovníku je komprese rychlejší, neboť dochází pouze k jednomu učení a následné kompresi všech rovin. Výsledky při použití globálního slovníku budou probrány v následujících kapitolách.

Samotný globální slovník se dá vytvořit pomocí dvou způsobů. Kde první, méně výpočetně náročný způsob, vytvoří slovník pomocí prostřední roviny bioinformatického datasetu. Proč je tento způsob přípustný, bude dále vysvětleno v sekci 4.1. Druhou metodou, která je mnohem více univerzální, je vytvoření slovníku ze všech rovin, řezů daného datasetu. Tato technika je více paměťově náročná, neboť se do paměti nahrají všechna data rovin. Zároveň je tato metoda velmi vhodná pro superpočítače, které mají tuto paměť a výpočetní výkon k dispozici. Navíc je při této volby hojně využito paralelizace, která celý proces urychluje.

Program nabízí řadu možností, jak celý proces komprese ovládat. Základní volbou je typ kvantizace, která se má použít a velikost slovníku. Velikost slovníku je zadávána jako mocnina dvojky. Našim cílem jsou slovníky velikostí 4 až 256. Velikost slovníku nepřímo určuje kompresní poměr, kde s menším slovníkem můžeme očekávat menší soubory, ale také větší chybu v obraze, která vznikne kvantizací. Při vektorové kvantizaci se také musí zvolit velikost kvantizačního vektoru, který je většinou dvoudimenzionální a můžeme si jej představit jako matici. Další možností je řádkový vektor. Jelikož jsou části našeho programu paralelizovány, tak si také můžeme zvolit počet jader, které chceme využít pro trénování slovníku, které celkově zabírá nejvíc času.

Účel dekompresního algoritmu je jasný, jeho hlavním úkolem je přečíst komprimovaný proud dat a převést jej na data obrazová. Mimo to umí také prozkoumat komprimovaný soubor a vypsat o něm informace. Operace dekomprese nepotřebuje žádné další volby od uživatele, neboť všechny důležité informace jsou uloženy v QCMP hlavičce. Podle hlavičky si vytvoří správný

dekompresor, který si již načte slovník a Huffmanův strom. Pomocí Huffmanova stromu jsou jednotlivé binární kódy dekódovány na indexy slovníkových záznamů. Slovníkové objekty jsou následně kopírovány do výsledného souboru. Obecně se slovník, spolu s informací, jak vybudovat Huffmanův strom, nachází před samotnými daty obrazové roviny. Pokud je použit globální slovník, tak ten je uložen před první rovinu a dále následují pouze obrazová data.

4 Analýza komprese pomocí kvantizačních technik

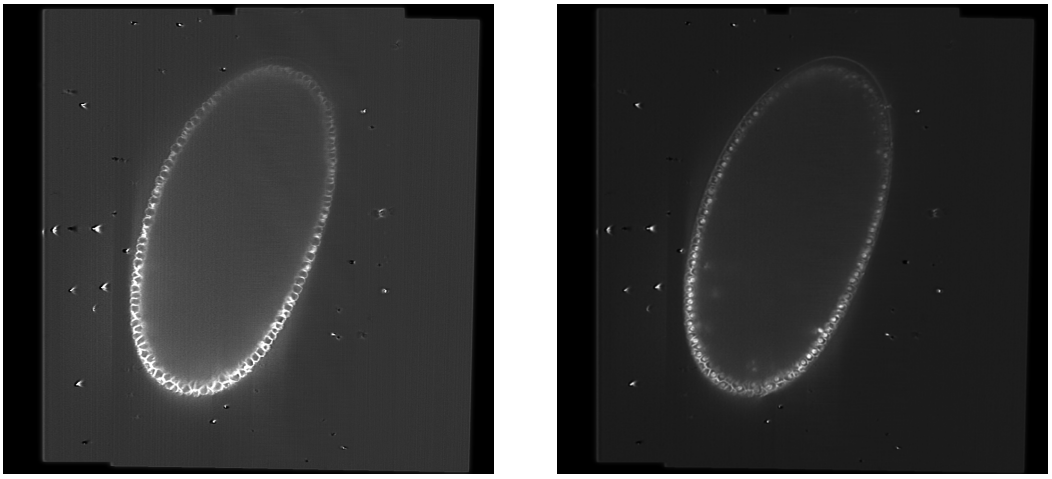
Cílem této kapitoly je seznámit čtenáře s výsledky, kterých jsme dosáhli. Provedeme detailní analýzu kompresních výsledků, které jsme získali použitím námi implementovaných algoritmů skalární a vektorové kvantizace. Než se dostaneme k samotným výsledkům, uvedeme a popíšeme na jakých bioinformatických datech budeme kompresi zkoušet. Spolu s představením testovaných dat, zmíníme některé společné charakteristiky bioinformatických datasetů. Pak již bude následovat analýza výsledků našich dvou algoritmů, nejdříve skalární kvantizace a následně vektorová kvantizace. Obě tyto metody nejdříve probereme zvlášť a poté je srovnáme mezi sebou i s kompresí JPEG2000.

4.1 Popis testovaného datasetu

Kompresi budeme testovat na datasetech získaných pomocí *Light Sheet* mikroskopie. V této metodě fluorescenční mikroskopie je vzorek osvětlován excitačním paprskem, vytvarovaným do plochy a procházející vzorkem kolmo k objektivu mikroskopu. Vždy je osvětlena pouze tenká vrstva vzorku, za vzniku roviny neboli řezu. Mikroskop se vždy po nasnímání jedné roviny posune o určitou vzdálenost, a takto nasnímá celý vzorek v několika rovinách. Ze vzorků získaných v jednom časovém bodu, lze poté také vytvořit 3D volumetrický obraz. Seskupení jednotlivých rovin nazýváme datasetem. Samotné snímání se dá provádět taky v čase, díky čehož se mohou vytvářet časosběrná videa vývoje vzorku. V jednom datasetu se proto obvykle nacházejí snímané roviny v několika časových bodech.

My máme pro zkoumání k dispozici dva datasety embrya octomilky (*Drosophila*), které byly získány ve stejný časový bod. Buňky embrya jsou geneticky modifikovány a jejich jednotlivé části mají na sobě napojené fluorescenční markery. Tyto markery reagují na různé barvy světla (různé vlnové délky) jinak. Celý vzorek je poté snímám pomocí dvou různých excitačních paprsků, lišících se ve vlnových délkách, za vzniku dvou datasetů. Ukázku z obou datasetů můžeme vidět na Obrázku 14, kde datasety rozlišujeme podle kanálu 0 a 1. Všechny roviny jsou rozměrů 1041×996 a v obou datasetech je jich celkem 946. Na těchto obrázcích si taky můžeme všimnout markerů mimo vzorky, které slouží k následné rekonstrukci 3D obrazu. Originálně byly tyto datasety uloženy pomocí 32 bitů na pixel. My jsme je pro naše účely převedli na 16 bitů na pixel, bez ztráty na kvalitě.

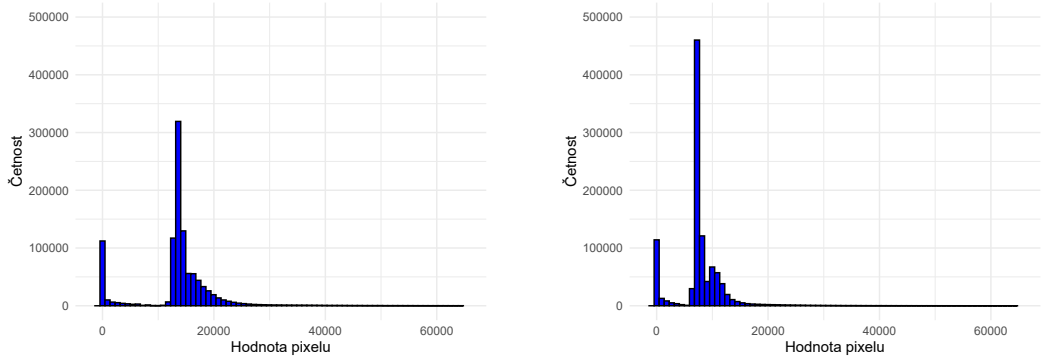
Dále na grafech v Obrázku 15 můžeme vidět rozdělení hodnot pixelů, které odpovídá snímkům z Obrázku 14. U obou kanálů platí, že i když je možný rozsah hodnot velký, tak většina pixelů má hodnotu do 30000. Samotný vzorek je v obrazu reprezentován světlejšími pixely, které mají větší hodnoty a vznikly reakcí na procházející světelný paprsek. Zároveň se velká část pixelů pohybuje blízko nuly. Toto je pravda obecně, neboť většinovou část obrazu zaujímá okolí, prostředí, ve kterém se vzorek nachází. Toto okolí nereaguje na světlo, a proto je tato část obrazu tmavá, reprezentována pixely s malými hodnotami. Informace získané z těchto histogramů



(a) Kanál 0 (b) Kanál 1

Obrázek 14: Ukázka z datasetu embrya octomilky

nám sdělují, že mnoho pixelů nabývá podobné hodnoty a mělo být možné vytvořit kvantizační objekty, které by dobře reprezentovaly komprimovaný obraz.



(a) Kanál 0 (b) Kanál 1

Obrázek 15: Histogram hodnot pixelů v datasetu embrya octomilky

Pří zmínce o globálním slovníku jsme slíbili, že v této sekci uvedeme vysvětlení, proč je možné využít prostřední roviny jako referenční. Referenční rovina je použita k natrénování slovníku pro všechny ostatní řezy. Z principu *Light Sheet* mikroskopie víme, že vzorek je snímán po řezech a můžeme tedy předpokládat, že v prostřední rovině bude vzorek přítomen. Toto nemusí, a v našem případě ani není, pravda pro první a poslední snímané roviny. V testovaných datasetech se vzorek objevuje až po 180. rovině a znovu mizí někdy po 720. rovině. Přítomnost vzorku v referenčním rovině je velmi důležitá. Pro kvantizační slovník je mnohem důležitější se naučit hodnoty pixelů, které reprezentují vzorek nežli pixely, reprezentující okolí vzorku. Okolí pro nás nenese důležitou informaci. Z tohoto důvodu předpokládáme, že prostřední řez obsahuje vzorek a naučený slovník bude obsahovat skaláry nebo vektory, které dokážou dobře zrekonstruovat

vzorek i v jiných rovinách. Toto by se dalo tvrdit i o jiných rovinách kolem prostřední, ale zrovna u této vybrané přepokládáme největší pravděpodobnost výskytu vzorku.

Co se týče entropie vstupních dat, tak tu najdeme v přílohách na Obrázku 30. V těchto dvou grafech můžeme sledovat entropii jednotlivých rovin pro oba testované datasety. Entropií v našem případě rozumíme minimální počet bitů, nutný k zakódování jednoho symbolu. Symboly jsou pro nás unikátní hodnoty pixelů, které se nacházejí v dané rovině datasetu. Na grafech si všimneme, že hodnota entropie je u obou datasetů velmi podobná. Dále vidíme, že entropie postupně roste na hodnotu více jak 11 bitů a následně zase klesá. Toto souvisí s tím, jak se biologický vzorek postupně objevuje na snímcích. Přítomnost vzorku znamená více různých hodnot jasu, a tak složitější informaci ke kompresi. V následujících sekcích si budeme popisovat grafy MAE pro jednotlivé řezy. Můžeme si všimnout, že hodnoty MAE budou růst podobně jako entropie řezů.

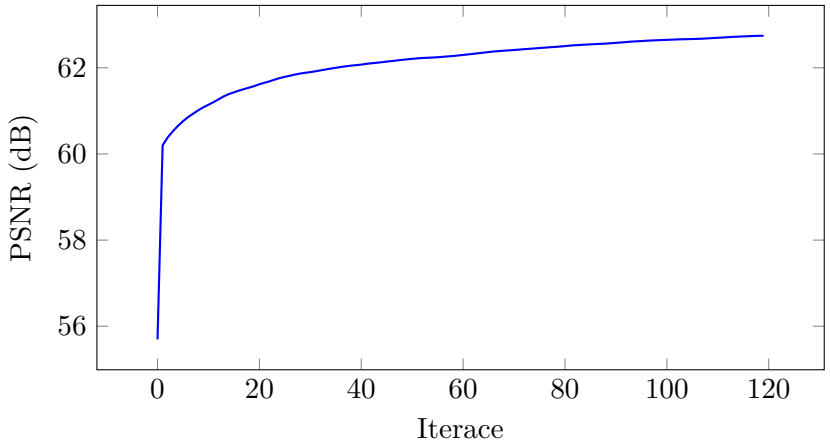
4.2 Výsledky skalární kvantizace

Zde se již dostáváme k samotným výsledkům, kterých jsme dosáhli použitím námi vytvořeného kompresního nástroje. V této sekci projdeme výsledky první metody, kterou je skalární kvantizace. Jak jsme již uvedli dříve, kompresní poměr a výsledná kvalita obrazu je řízena velikostí slovníku. Vyzkoušíme zde slovníky sedmi různých velikostí od $L = 4$ až po $L = 256$. Od nejmenšího slovníku s pouze čtyřmi položkami neočekáváme žádné dobré výsledky, ale budeme jej uvádět pro celkové srovnání. Slovník s více než 256 objekty jsme již nezkoušeli, neboť bychom potřebovali více než 8 bitů na zakódování jednoho indexu do slovníku. Počet bitů na zakódování jednoho indexu označujeme pomocí n. Kdybychom nepoužívali Huffmanovo kódování, které dále mění počet bitů na index, tak by se dal kompresní poměr skalární kvantizace vypočítat podle rovnice 11. Kde W značí šířku obrazu a H jeho výšku. Tato rovnice počítá s uložením slovníku spolu s každým řezem a také s 16 bity na jeden pixel. Je tedy zřejmé, že při použití 8 bitů na pixel ($L = 256$) bychom měli dosáhnout $CR_{sq} \approx 0,5$.

$$\mathrm{CR}_{sq} = \frac{16L + (nWH)}{16WH} \tag{11}$$

V grafu na Obrázku 16 můžeme vidět průběh trénování optimálního slovníku $L = 256$. Průběh je ilustrován měnící se hodnotou PSNR. Tato veličina je již na počátku poměrně dobrá, čemuž vděčíme naší úpravě Lloyd–Max algoritmu. Celkově PSNR roste nejvíce ze začátku a s rostoucími iteracemi je zlepšení méně významné. Podobně to platí pro pokles chyby MSE.

Výsledky skalární kvantizace bez použití Huffmanova kódování pro stejný snímek, jako je na Obrázku 14a najdeme v Tabulce 7. Můžeme potvrdit, že s větším slovníkem klesá vzniklá chyba a zároveň roste hodnota BPP. Při výpočtu obou hodnot CR_{sq} i BPP je do velikosti výsledného souboru započítávána velikost slovníku. Toto je důvodem, proč není hodnota BPP celočíselná a odchyluje se od n. Větší slovníky pak zabírají více místa, a proto u nich pozorujeme větší



Obrázek 16: PSNR v průběhu iterací Lloyd–Max algoritmu

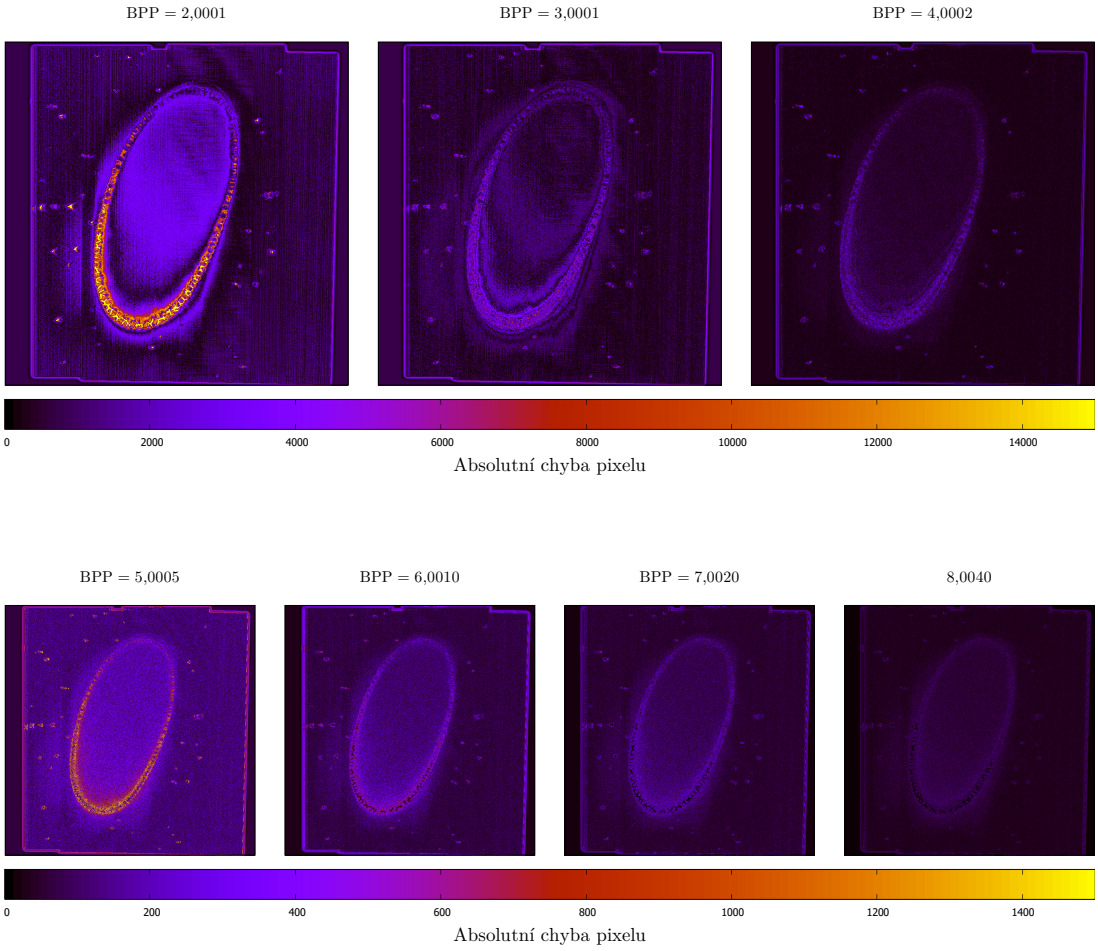
n	L	MSE	PSNR (dB)	CR_{sq}	BPP
2	4	5275546,0477	29,1068	0,1250	2,0001
3	8	1214174,9001	35,4867	0,1875	3,0001
4	16	305193,0568	41,4837	0,2500	4,0002
5	32	73796,0319	47,6491	0,3125	5,0005
6	64	19260,3213	53,4828	0,3751	6,0010
7	128	7078,7052	57,8299	0,4376	7,0020
8	256	2282,6575	62,7451	0,5002	8,0040

Tabulka 7: Výsledky skalární kvantizace pro embryo octomilky (Kanál 0)

odchylku od n. Vliv Huffmanova kódování na výsledný kompresní poměr bude uveden později. Je ale jisté, že s jeho použitím dosáhneme pouze zlepšení.

Samotná čísla uvedené v předcházející tabulce, nám sami o sobě moc neřeknou, a proto na Obrázku 17 vidíme absolutní chybu pixelu přímo na komprimovaném obrazu. Nutno podotknout, že obrázky v první a druhé řadě mají různou škálu chyby. Na těchto obrázcích vidíme postupně vývoj vzniklého kvantizačního šumu. Ze začátku jsou nejvíce chybné pixely samotného snímaného vzorku. Toto je ale případ s extrémně malým slovníkem. Postupně se chyba pixelů, které vyobrazují vnitřní strukturu buňky zmenšuje a většina výrazných chyb se přesunuje do okolí vzorku. V okolí vzorku se chyba převážně vyskytuje na markerech, které jsou důležité pro automatizovanou rekonstrukci 3D obrazu. Tyto markery, však nejsou příliš významné pro člověka, který se dívá hlavně na vzorek.

U prvních dvou z těchto obrázků ihned vidíme výraznou chybu, způsobenou malým rozsahem možných hodnot. Avšak dále, od třetího obrázku, se již chyba hledá hůře. Vidíme, že struktura vzorku je správně zachována a vizuální chyba se vyskytuje hlavně v přechodu mezi vzorkem a jeho nitrem. Co je hlavní, nedochází ke vzniku žádných vizuálních artefaktů. V přílohách na Obrázcích 24 a 25 najdeme již přímo výsledné obrázky. U těchto obrázků je již hodnota BPP vypočtena s Huffmanovým kódováním a je vynechán nejmenší slovník $L = 4$.

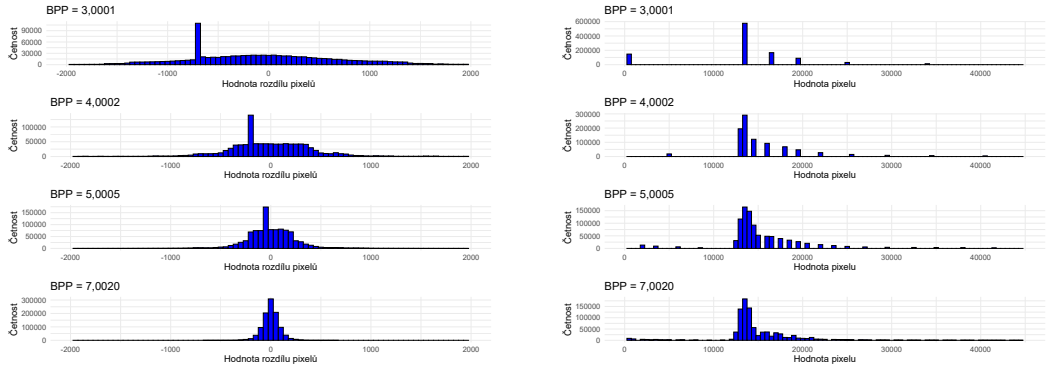


Obrázek 17: Absolutní chyba ve vybraném řezu

Dále bychom se více podívali na samotnou kvantizační chybu. Vybrané statistické charakteristiky této chyby pro jednotlivé velikosti slovníku můžeme najít v Tabulce 8. Z této tabulky vidíme, že rozdělení chyb u všech slovníku je pozitivně zešikmené a zároveň velmi špičaté. Normálnímu rozdělení se v rámci špičatosti blíží pouze poslední slovník $L = 256$. Dle vývoje mediánu můžeme pozorovat, jak se celá distribuce postupně posouvá na pravou stranu blíže k nule. Samotné zužování histogramu a přibližování se tak k Diracovu impulsu, lze dobře vyčíst ze snižující se hodnoty směrodatné odchylky. V naprosto ideálním případě by distribuce chyby vypadalo jako Diracův impuls. Všechny hodnoty chyby by byly nulové. Toto zužování histogramu chyby můžeme graficky pozorovat na Obrázku 18a. Následně na Obrázku 18b vidíme distribuci hodnot pixelů po provedení komprese. Tyto histogramy můžeme porovnat s histogramy na Obrázku 15a. Distribuce hodnot pixelů po kompresi by se měla co nejvíce blížit originálnímu rozdělení hodnot jasu, toto při porovnání grafů můžeme potvrdit.

n	L	Průměr	Medián	Směr. odch.	Šikmost	Špičatost
2	4	0,3372	-501	2296,8568	1,8731	18,7650
3	8	0,6676	-99	1101,8964	0,7427	11,6360
4	16	0,4885	-46	552,4429	0,4871	12,3575
5	32	0,1379	-25	271,6543	0,0781	10,3603
6	64	0,4409	-13	138,7809	0,1836	8,3237
7	128	0,4291	-8	84,1340	0,0502	5,4406
8	256	0,5416	-4	47,7741	0,0284	3,5307

Tabulka 8: Výběrové charakteristiky hodnoty chyby skalární kvantizace



(a) Histogram vzniklé kvantizační chyby (b) Histogram pixelů po skalární kvantizaci

Obrázek 18: Distribuce hodnot v obraze po skalární kvantizaci

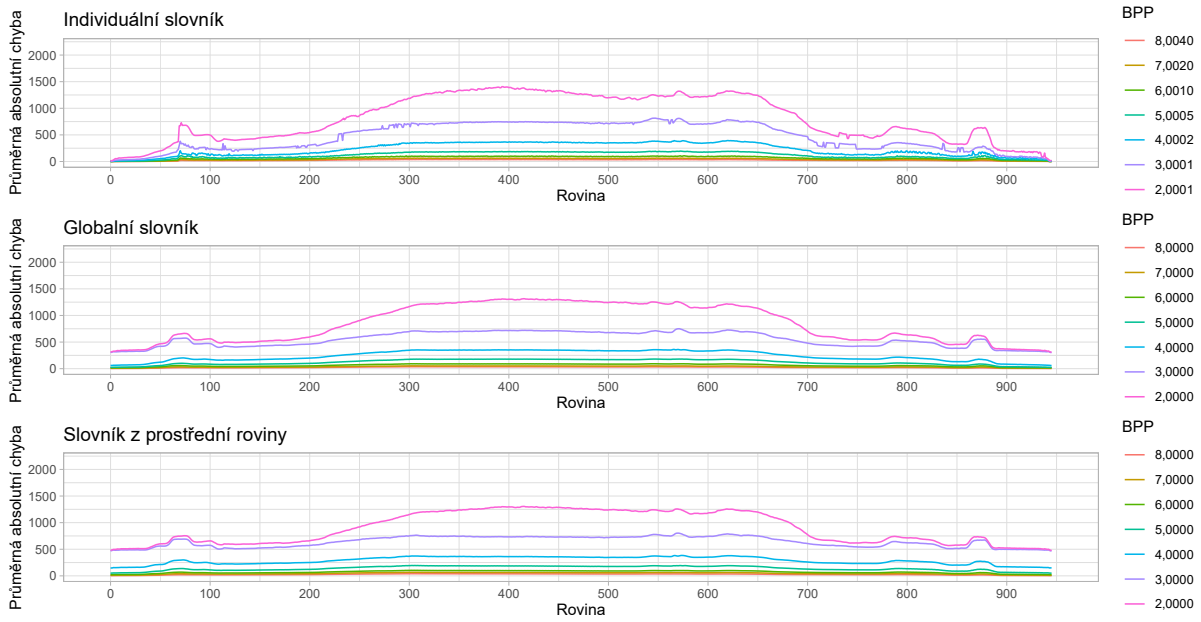
Co se týče normality kvantizační chyby, tak ta může být dle vybraných statistických charakteristik v Tabulce 8 zavrhnuta. Hlavním problémem je velká špičatost. Toto tvrzení si potvrdíme provedením Kolmogorovova–Smirnovova testu. Shapiro–Wilkův test nemohl byt použit, neboť velikost výběrů přesahuje limit 5000 vzorků. Nulová hypotéza H_0 říká, že data pocházejí z normálního rozdělení. Alternativní hypotéza je rovna negaci nulové hypotézy, $H_A = \neg H_0$. p-hodnota pro daný test vyšla $\ll 0,0001$. Na hladině významnosti 5 % tedy zamítáme nulovou hypotézu ve prospěch alternativní hypotézy a potvrzujeme, že rozdělení kvantizační chyby není normálním rozdělením.

Do této doby jsme se bavili pouze o kompresi jednoho vybraného řezu, z kanálu 0 datasetu embrya octomilky. V dalším grafu této sekce se podíváme a probereme výsledky, kterých jsme dosáhli kompresí všech rovin obou datasetů. V grafech na Obrázku 18 můžeme vidět vývoj průměrné absolutní chyby pixelu MAE vzhledem k rovině datasetu. MAE je vypočítáno podobně jako MSE, akorát místo sumy druhé mocniny rozdílu je sčítána absolutní hodnota rozdílu.

Zároveň na těchto grafech porovnáváme tři různé typy slovníků. V prvním grafu je každá rovina komprimována podle svého vlastního slovníku. U nejmenšího slovníku $L = 4$ můžeme pozorovat kolísání MAE, neboť 4 hodnoty nestačí ke kompresi detailního obrazu. Toto kolísání můžeme ještě dramatičtěji pozorovat pro výsledky z druhého kanálu. Grafy pro tento druhý

kanál najdeme na Obrázku 31 v přílohách. Na tomto grafu si všimneme velmi divného stoupání chyby od cca 290. roviny. Právě někdy od této roviny začíná okolí vzorku zabírat významnou část obrazu. Toto okolí je velmi tmavé, obsahující pouze pixely malých hodnot. Avšak většinová část obrazu je stále zabírána světlejšími pixely, a proto algoritmus vytváří slovník, obsahující právě tyto vyšší hodnoty. Se zvětšováním se tohoto tmavého okolí roste vzniklá chyba MAE. Stejná situace nastává pro roviny 550 až 560. Dle grafu vidíme, že toto je pouze chyba nejmenšího slovníku.

Zbylé dva grafy reprezentují výsledky dosáhnuté pomocí jediného slovníku. Tento slovník byl využit pro kompresi všech rovin v datasetu. Ihned si všimneme, že tyto výsledky nejsou pro žádnou velikost slovníku horší, než tomu bylo v prvním grafu. Je tomu právě naopak, křivky vypadají stabilněji. Například v Obrázku 31 vidíme, že nedochází ke vzniku velké chyby na několika za sebou jdoucích rovinách. Velmi dobré výsledky slovníku vytvořeného z prostřední roviny potvrzují naši hypotézu, že uprostřed datasetu se nachází řez, obsahující snímaný vzorek. Díky přítomnosti vzorku slovník obsahuje správné kvantizační hodnoty, hodící se k rekonstrukci obrazu. Toto, avšak nemusí být vždy pravda, a proto je jistější využít slovník globální, který je vytvořen ze všech rovin datasetu. Jedinou nevýhodou globálního slovníku, oproti slovníku z prostřední roviny je délka učení. V přílohách na Obrázku 32 najdeme stejný graf hodnoty MAE podle rovin, avšak pouze pro 4 nejlepší slovníky. Na těchto přiložených grafech můžeme detailněji pozorovat výsledky lepších slovníků.



Obrázek 19: Průměrná absolutní chyba skalární kvantizace podle rovin (Kanál 0)

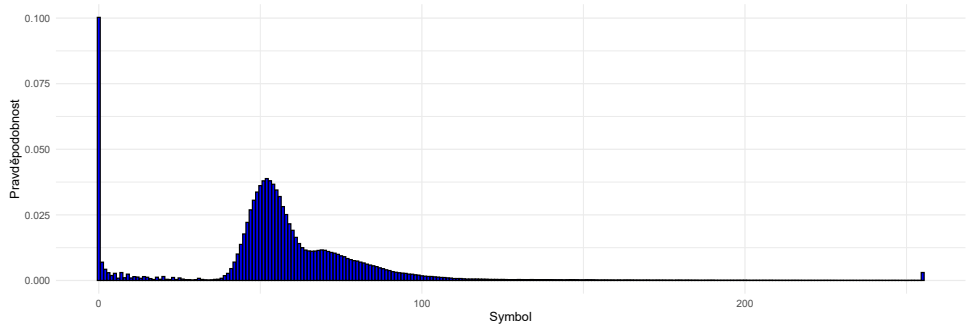
Globální slovník s sebou přináší tři velké výhody. Zaprvé je naučen pouze jednou a uložen do *cache* souboru. Tato operace je sice náročnější, ale jelikož počítáme s využitím na superpočítačích tak pro nás to není problém. Zadruhé rychlost komprese je mnohem vyšší. Slovník se

načte z *cache* souboru a trénovací krok kompresoru se může přeskočit. Čas dekomprese je taky kratší, neboť se slovník načítá z komprimovaného souboru pouze jednou. Třetí výhodou je menší komprimovaný soubor a s tím i lepší kompresní poměr, protože se v souboru se nachází pouze jeden slovník.

Dosud jsme ve výsledcích pracovali s kompresním poměrem vypočteným podle Rovnice 11. Avšak v sekci 3.5 jsme uvedli využití Huffmanova kódování. Jak toto kódování ovlivnilo výsledný kompresní poměr je uvedeno v Tabulce 9. V této tabulce, ve sloupci CR, uvádíme kompresní poměr s Huffmanovým kódováním a ΔCR je zlepšení oproti kompresi bez tohoto kódování. Stejné platí i pro sloupec BPP. Díky tomuto přidanému kroku v rámci komprese jsme schopni ušetřit až 2 bity na pixel, jak je tomu u slovníku s $L = 256$. Celkově je kompresní poměr po využití Huffmanova kódování lepší. Pravděpodobnosti jednotlivých symbolů, pomocí kterých je tvořen Huffmanův strom můžeme, vidět v grafu na Obrázku 20. V tomto grafu má symbol 0 největší pravděpodobnost, takže mu bude přiřazen nejkratší binární kód. Díky této informaci o pravděpodobnosti symbolů taky víme, jak využívané jsou jednotlivé kvantizační hodnoty ve slovníku.

n	L	CR	ΔCR	BPP	ΔBPP
2	4	0,0898	0,0352	1,4367	0,5634
3	8	0,1221	0,0654	1,9541	1,0460
4	16	0,1909	0,0591	3,0538	0,9464
5	32	0,2454	0,0671	3,9268	1,0737
6	64	0,3002	0,0749	4,8027	1,1983
7	128	0,3299	0,1077	5,2784	1,7236
8	256	0,3719	0,1283	5,9502	2,0538

Tabulka 9: Kompresní poměr skalární kvantizace s Huffmanovým kódováním



Obrázek 20: Pravděpodobnosti symbolů

4.3 Výsledky vektorové kvantizace

V této kapitole se budeme zabývat výsledky vektorové kvantizace, které je zobecněním skalární. Problémem skalární kvantizace je omezený kompresní poměr, kterého lze dosáhnout kompresí jednotlivých pixelů. Proto vektorová kvantizace provádí kompresi po blocích pixelů. Z tohoto důvodu budeme dosahovat lepších kompresních poměrů, za cenu větší kvantizační chyby. Pro maximalizaci kompresního poměru bychom mohli využívat velkých kvantizačních bloků, které by avšak vedly k nadmíru znatelným chybám. Úkolem je tedy vybrat správně velký blok, který nám umožní vytvořit co nejmenší soubory s minimálně znatelnou vizuální chybou.

Co se týče počtu a velikosti slovníků, tak platí stejné informace, které byly uvedeny v přecházející sekci 4.2. První rozdíl je v Rovnici 12, pomocí které se dá vypočítat kompresní poměr vektorové kvantizace bez využití Huffmanova kódování. Tato rovnice platí pro dvoudimenzionální obrazy, kde kvantizační vektory je široké v_x a vysoké v_y.

$$\mathrm{CR}_{vq} = \frac{16L(v_x v_y) + n\left(\lceil W/v_x \rceil \lceil H/v_y \rceil\right)}{16WH} \tag{12}$$

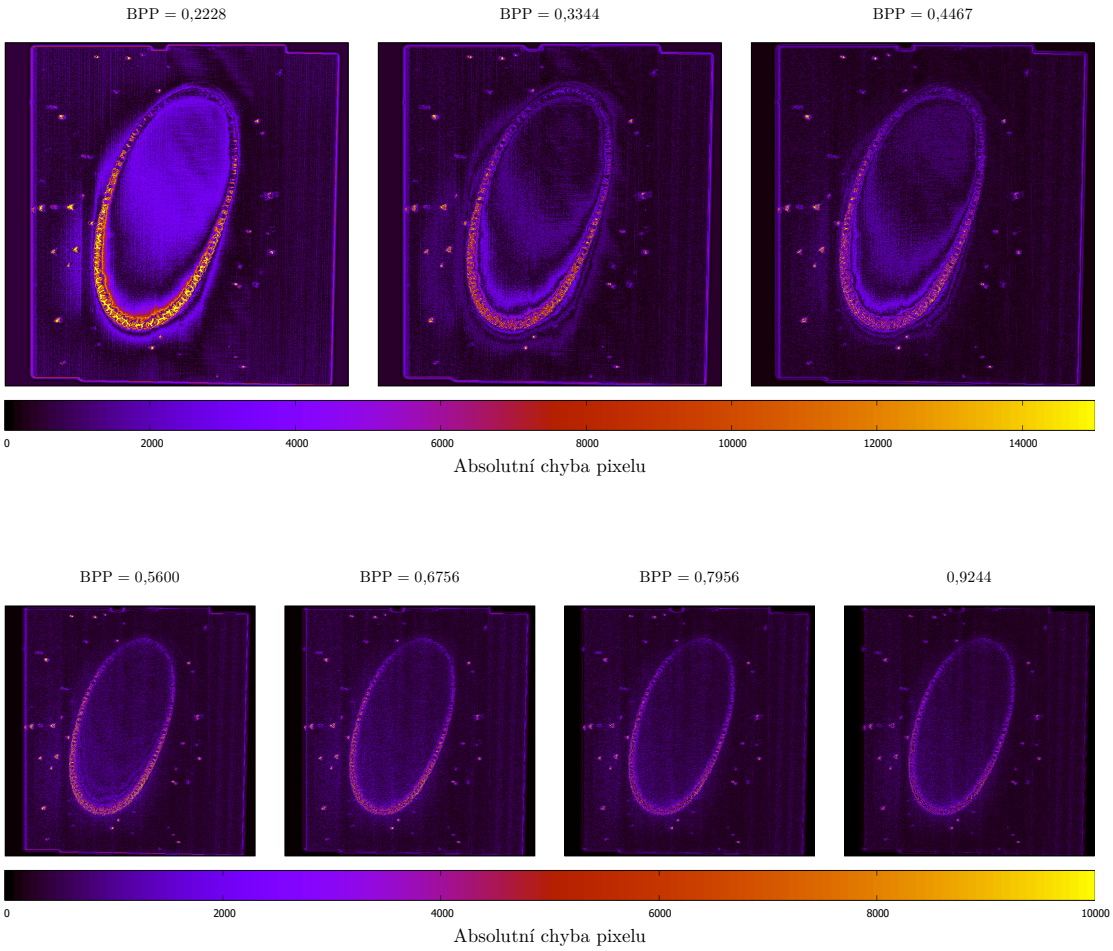
V této práci jsme se zaměřili hlavně na kompresi pomocí vektorů velikosti 9, $N = 9$. Jeden objekt slovníku tedy reprezentuje celkem 9 pixelů v obraze. Vyzkoušeli jsme řádkový vektor rozměrů 9×1 a maticový vektor 3×3. Výsledky vektorové kvantizace provedené s vektorem 3×3 na vybraném řezu nalezneme v Tabulce 10. Tyto výsledky můžeme srovnat s hodnotami v Tabulce 7.

L	MSE	PSNR (dB)	CR_{vq}	BPP
4	6419376,3469	28,2545	0,0139	0,2228
8	2786779,4970	31,8784	0,0209	0,3344
16	1959095,5248	33,4089	0,0279	0,4467
32	1271817,9621	35,2852	0,0350	0,5600
64	911822,9665	36,7304	0,0422	0,6756
128	627429,8214	38,3538	0,0497	0,7956
256	471523,5292	39,5944	0,0578	0,9244

Tabulka 10: Výsledky vektorové kvantizace 3×3 pro embryo octomilky (Kanál 0)

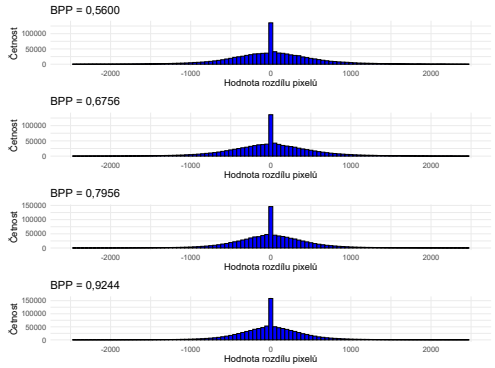
Dle této tabulky se zdá, že kvalita výsledného obrazu je po provedení vektorové kvantizace velmi špatná. PSNR dosahuje v nejlepším případě pouze hodnoty 39,5944 dB, kdežto u skalární kvantizace jsme se dostali až na hodnotu 62,7451 dB. Stejně tak chyba MSE je mnohonásobně vyšší. Na obrázcích, které budou následovat, si vysvětlíme, proč došlo ke vzniku tak velkých chyb. Ještě v této tabulce si všimneme velmi pozitivního posledního sloupce. U všech slovníků jsme se dostali na hodnotu menší než 1 bit na pixel. Vysvětlení nízké hodnoty BPP je jednoduché. Jak jsme již zmínili, jeden blok zastupuje N pixelů a k zakódování obrazu je potřeba mnohem méně bloků, než je pixelů.

Abychom lépe porozuměli chybě, která vzniká vektorovou kvantizací, podíváme se znova na teplotní mapy, které najdeme na Obrázku 21. Na prvních třech grafech vidíme, že většinová část chybných pixelů se vyskytuje přímo na buněčné stěně vzorku. Navíc se tyto pixely liší o maximální hodnotu teplotní mapy, někdy i víc jak 15 000. Z tohoto můžeme usoudit, že pro aplikaci vektorové kvantizace nebude možné využít slovníku menších nežli 32. Na dalších grafech vidíme, že chyba na pixelech vzorku klesá a teplejší místa se nacházejí spíše v okolí vzorku. Tyto chybné pixely na markerech jsou taky důvodem velmi vysokých hodnot MSE a malé hodnoty PSNR v předcházející tabulce.

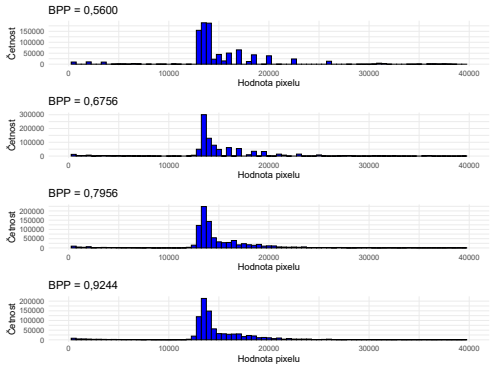


Obrázek 21: Absolutní chyba ve vybraném řezu vektorová kvantizace

Obecně jsou tyto teplotní mapy více zašuměné, něž tomu bylo u skalární kvantizace. U té byl pixel nahrazován jedním pixelem, kdežto zde je nahrazován v rámci bloku. Stává se tedy, že i když je slovníkový vektor nejblíže bloku obrazu, tak některý pixel nebo pixely mohou být zcela rozlišné. Tento efekt se dá ještě více pozorovat na řádkovém vektoru. U něj si můžeme představit, že se budou bloky v prvních $N - 2$ pixelech rovnat a zbylé 2 pixely budou odlišné.



(a) Histogram vzniklé kvantizační chyby, vektorová kvantizace



(b) Histogram pixelů po vektorové kvantizaci

Obrázek 22: Distribuce hodnot v obraze po vektorové kvantizaci

Díky tomuto jevu bohužel pozorujeme několik velkých chyb v rámci buněčné stěny embrya octomilky. Avšak v posledním grafu se již největší chyby vyskytují v okolí vzorku na markerech. Tyto objekty nejsou pro člověka důležité, a tudíž se dá tato chyba tolerovat. V budoucí práci by stálo za vyzkoušení provést předzpracování obrazu, které by odstranilo okolí vzorku od vzorku samotného. Samotné snímky po dekompresi najdeme znovu v přílohách na Obrázcích 28 a 29. Při typickém prohlížení obrázku je poměrně náročné vidět vizuální artefakty vzniklé kompresí. Samozřejmě kvalita by se dala zlepšit větším slovníkem nebo menším kvantizačním vektorem.

Histogramem na Obrázku 22a si můžeme potvrdit, že pixely lišící se výrazně od původních pixelů zaujímají menšinovou část v obraze. Nejčetnější jsou opět hodnoty v okolí nuly, tedy minimální rozdíly. Oproti skalární kvantizaci je rozsah chyby v okolí nuly širší. Toto je znovu způsobeno efektem, který jsme zmínili v předcházejícím odstavci. Tento efekt má za důsledek vznik většího počtu malých chyb. Na vedlejším histogramu v Obrázku 22b znovu vidíme distribuci hodnot pixelů po dekompresi. Tyto histogramy už nebudou mít přesně L vrcholů, jak tomu bylo u skalární kvantizace. Což může být výhodou vektorové kvantizace, která je schopna zrekonstruovat více než jen L hodnot jasu. Toto platí díky slovníku, obsahujícího obecně více unikátních hodnot.

Porovnáme-li statistické výběrové charakteristiky vektorové kvantizace v Tabulce 11, oproti skalární kvantizaci v Tabulce 8 najdeme několik významných rozdílu. Tyto odlišnosti dále potvrzují naše předchozí tvrzení. Významně větší směrodatná odchylka nám potvrzuje dříve zmíněný širší rozsah hodnot a bohužel taky říká, že tato metoda bude hůře dosahovat ideálního Diracova impulsu. Co se týče tvaru histogramu chyby, tak ten je znovu pozitivně zešikmený, ale jeho špičatost je výrazně vyšší. Jelikož se chyba pixelu počítá rozdílem kvantizovaného pixelu od originálního pixelu, tak kladná hodnota průměru znamená, že běžně je pixel po kvantizaci tmavší. Díky tomu můžeme říct, že při kvantizaci dochází k určité ztrátě jasu, která je avšak při větších slovnících nevýznamná. O normalitě chyby už v tomto případě ani neuvažujeme, a to hlavně díky vysoké špičatosti.

L	Průměr	Medián	Směr. odch.	Šikmost	Špičatost
4	30,7036	-393	2533,4640	1,9868	32,2008
8	36,0887	-103	1668,9757	1,1205	80,5539
16	48,5230	-46	1398,8363	0,5664	97,0722
32	13,0592	-17	1127,6740	0,4302	117,9198
64	1,7319	-5	954,8931	-0,0966	111,4976
128	4,7748	-1	792,0907	-0,0065	112,6596
256	3,9256	0	686,6648	0,0519	91,8357

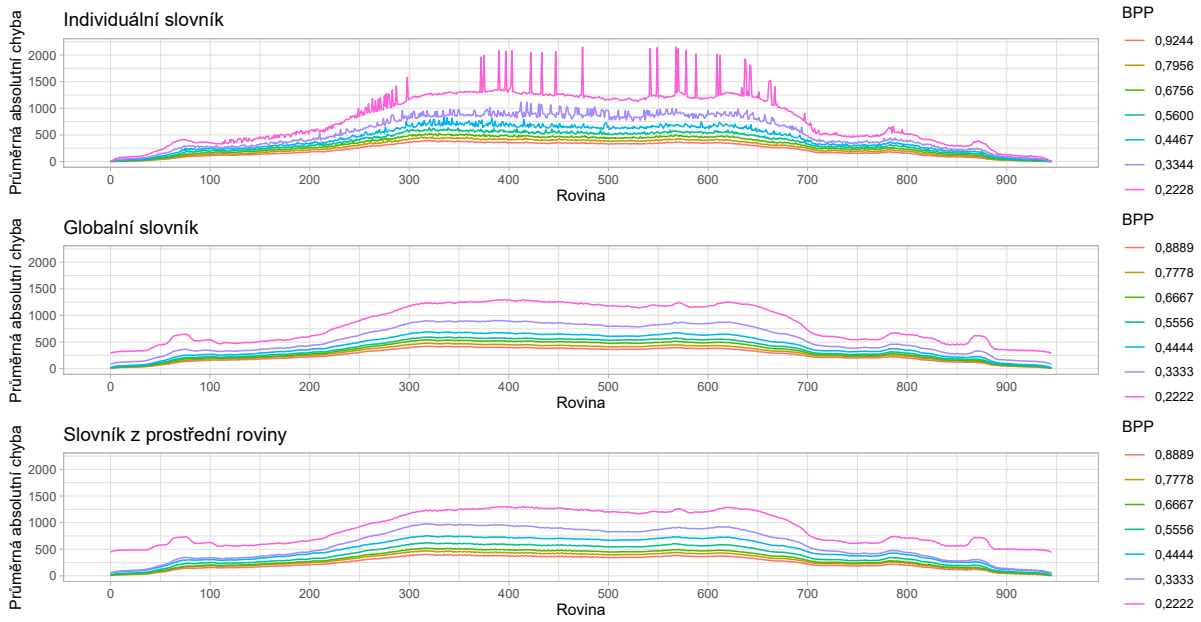
Tabulka 11: Výběrové charakteristiky hodnoty chyby vektorové kvantizace

Dále si podobně jako v předchozí kapitole probereme průměrné absolutní chyby v rámci všech rovin datasetu. Také srovnáme výsledky individuálního slovníku se slovníkem globálním. Budeme se zabývat kvantizačním vektorem s rozměrem 3×3, který lépe využívá vlastností obrazů. Grafy MAE pro kanál 0 nalezneme na Obrázku 23 a pro kanál 1 v přílohách na Obrázku 33. Prvního, čeho si všimneme oproti skalární kvantizaci je četný výkyv chyby pro nejmenší slovník. Znovu proto potvrzujeme, že 4 různé vektory nestačí na zakódování obrazu, výsledný obraz je nepoužitelný. Hlavní problém je ve tvorbě slovníku, kdy se někdy podaří vybrat 4 poměrně dobře reprezentující vektory, avšak v jiných případech budou převládat vektory, které špatně reprezentují komprimovaný obraz. Tyto výkyvy jsou ještě výraznější na grafu v Obrázku 33. Obecně platí, že čím je slovník větší, tím je průměrná hodnota chyby podle rovin stabilnější.

Při porovnání si všimneme, že ve všech třech grafech je průměrná chyba nejlepšího slovníku vždy větší, než tomu bylo u skalární kvantizace. Pozitivní je, že globální slovník se znovu ukazuje jako velmi silná technika komprese. Výsledky tohoto typu slovníku jsou nejstabilnější a také nejlepší. Navíc v kanálu 1 je chyba ještě menší. Hodnoty BPP uvedeny v těchto grafech jsou vypočteny bez použití Huffmanova kódování. Dále u prvního grafu jsou BPP větší, protože v komprimovaném souboru je uloženo 946 slovníků, oproti jednomu slovníku ve dvou následujících grafech.

Samozřejmě i ve vektorové kvantizaci jsme využili Huffmanovo kódování, které opět vedlo ke zlepšení obou parametrů CR i BPP. U této metody ale není vylepšení tak výrazné, jak u metody předcházející. Kompresní poměry spolu s rozdílem najdeme v Tabulce 12. Kompresní poměr vektorové kvantizace je už tak velmi dobrý, takže i zlepšení o setiny, jak tomu je u větších slovníků, je velmi výhodné. U BPP jsme byli schopni snížit hodnotu průměrně o 0,15 bitů na pixel. Navíc při kompresi datasetu pomocí jednoho globálního slovníku očekáváme výraznější zlepšení BPP.

Na konci této sekce se ještě podíváme na propustnosti, rychlosti obou implementovaných metod, při použití globálního slovníku. Globální slovník jsme využili, protože jsme chtěli porovnat pouze časy komprese/dekomprese a nebrat v potaz čas potřebný pro naučení slovníku. Čas učení slovníku je výrazně delší u vektorové kvantizace. Výsledky byly změřeny na



Obrázek 23: Průměrná absolutní chyba vektorové kvantizace (3×3) podle rovin (Kanál 0)

L	CR	ΔCR	BPP	ΔBPP
4	0,0103	0,0036	0,1646	0,0582
8	0,0141	0,0068	0,2254	0,1090
16	0,0194	0,0085	0,3099	0,1368
32	0,0243	0,0107	0,3883	0,1717
64	0,0319	0,0103	0,5104	0,1652
128	0,0372	0,0125	0,5957	0,1999
256	0,0466	0,0112	0,7459	0,1785

Tabulka 12: Kompresní poměr vektorové kvantizace s Huffmanovým kódováním

počítači s procesor Intel Core i7-8850H, komprese i dekomprese byla prováděna sériově. Naměřené hodnoty jsou uvedeny v Tabulce 13. Pomocí SQ značíme skalární kvantizaci a pomocí VQ vektorovou kvantizaci. Dle naměřených hodnot zjišťujeme, že pro obě metody propustnost kompresoru klesá spolu s rostoucím slovníkem. Toto je logické, neboť kompresor musí vyhledat vhodnou slovníkovou hodnotu z většího slovníku, kterou nahradí originální pixel nebo blok pixelů. U vektorové kvantizace je toto zpomalení více dramatické a s každým dalším slovníkem je propustnost dvojnásobně menší. Toto je způsobeno větším počtem porovnání, kdy musí být pro každý vektor porovnáno N hodnot. Oproti tomu v dekompresi je vektorová kvantizace mnohem výkonnější a rychlost klesne až u větších slovníků. Pomalejší rychlost skalární kvantizace je daná metodou nahrazování pixelů postupně po jednom.

Nutno podotknout, že komprese i dekomprese jsou v aktuální době implementovány sériově a metody jsou poměrně snadno paralelizovatelné. Oba implementované algoritmy pracují při kompresi a dekompresi pouze s daty aktuální roviny a s konstantním slovníkem, který jim byl pře-

Metoda	L	Propustnost komprese (MB/s)	Propustnost dekomprese (MB/s)
SQ	4	146,3631	203,1817
	8	115,0506	145,5821
	16	74,3411	85,8877
	32	52,6046	58,8911
	64	42,9385	65,4982
	128	34,7163	46,0345
	125	25,6509	38,1008
VQ	4	82,1126	148,3101
	8	48,3193	148,7356
	16	28,5913	144,0050
	32	15,5943	142,1102
	64	8,1177	129,0721
	128	4,2009	119,6165
	125	2,1162	122,8333

Tabulka 13: Porovnání rychlostí skalární a vektorové kvantizace (globální slovník)

dán. Tyto výsledky byly získány zpracováním 250 rovin z datasetu. Můžeme si tedy představit, že při kompresi bude každé vlákno komprimovat každou rovinu zvlášť. Stejné platí pro dekompresi. Navíc je pro nás důležitější rychlost dekomprese, neboť počítáme s tím, že komprese bude prováděna na výkonném serveru nebo superpočítači. Zde mohou být obrazy kódovány za běhu, nebo už mohou být uloženy ve speciálním formátu QCMP. Komprimovaná data budou přenášena k uživateli, na jehož počítači bude docházet k dekompresi, která je již nenáročná a rychlá.

4.4 Porovnání s JPEG2000

V této poslední kapitole bychom rádi uvedli srovnání s kompresí JPEG2000, pro kterou jsme využili nástroj ImageMagick. Obecně se naše metody velmi liší od tohoto algoritmu, který je obecný a dá se použít na široké spektrum obrazů. Úroveň komprese jsme ve zvoleném nástroji řídili pomocí parametru `jp2:rate`. Hodnota tohoto parametru je opačná vzhledem k námi nadefinovanému CR. Zároveň jsme při použití tohoto nástroje nastavili zachování 16 bitů na pixel, tak jak je tomu u našeho programu. JPEG2000 nabízí vytvoření několika rozlišení při kompresi, my jsme vždy vytvořili pouze jedno. Výsledky metody JPEG2000 pro dva vybrané obrazy najdeme v přílohách v Tabulce 14. Kompresní poměr CR a BPP byly vypočteny z velikosti výsledného JPEG2000 souboru.

Jako prvního si všimneme, že JPEG2000 je schopen pomocí nejmenší úrovně komprese vytvořit kvalitnější obrazy, než je tomu u našich kvantizačních metod s největším slovníkem $L = 256$. Porovnáme-li poté výsledky skalární kvantizace, tak zjistíme, že JPEG2000 dokáže dosáhnout lepšího kompresního poměru a zároveň zachovat více původní obrazové informace. Dále se přesuneme na vektorovou kvantizaci, u které v základu očekáváme větší obrazovou chybu. Pokud

se podle dosažených kompresních poměrů podíváme na podobné výsledky obou metod, tak zjistíme, že jsou si hodnoty PSNR velmi blízké. Zároveň jsme schopni pomocí vektorové kvantizace dosáhnout menší hodnoty BPP při PSNR $\approx 36,7$.

Je důležité zde zmínit, že i když JPEG2000 může dosahovat lepších výsledků na jednom obraze, tak neobsahuje žádný mód, který by zpracovával všechny roviny datasetu. Toto je výhodou našeho nástroje, který dokáže zpracovávat celý dataset podle jednoho globálního slovníku. Můžeme tedy předpokládat, že při kompresi datasetu dosáhneme lepších výsledků, přesněji menšího kompresního poměru a rychlejší komprese.

5 Závěr

Cílem této práce bylo vyzkoušet techniky komprese obrazu, které by se dali aplikovat na bioinformatické datasety získané pomocí mikroskopických metod a technologií. Probrali jsme zde hlavně ztrátovou kompresi obrazu, ale uvedli jsme také bezztrátové techniky. V úvodu jsme zmínili důvody proč vůbec chceme provést kompresi obrazu. Jednalo se hlavně o zkrácení doby přenosu dat mezi počítači přes internetovou síť. Jeden z dalších důvodů byly menší nároky na uchovávání dat. Abychom byli schopni co nejvíce urychlit přenos bioinformatických datasetů mezi počítači, dali jsme si za úkol dosáhnout co nejlepších kompresních poměrů, a tedy co maximálně minimalizovat hodnotu BPP. Zároveň musíme brát v potaz kvalitu výsledného obrazu, protože poslední věc, kterou bychom chtěli je převést obraz do podoby, která by byla zcela nepoužitelná. Dalším nárokem, který jsme si uložili byla rychlá dekomprese. Rychlost dekomprese je důležitá, neboť tato operace je prováděna na straně klienta, který může mít méně výkonný počítač.

V první části tohoto dokumentu jsme se seznámili s existujícími technikami bezztrátové komprese. Jednalo se především o univerzální algoritmy a metody, které se dají aplikovat na všechny druhy dat. Také jsme v této části vyzkoušeli jeden algoritmus pro technologii CUDA, který navíc obsahoval mód pro ztrátovou kompresi. Z vyzkoušených metod jsme ohodnotili jako nejlepší algoritmus bzip2, který konzistentně dosahoval nejlepších kompresních poměrů a byl ze všech nejrychlejší. Co se týkalo rychlosti samotné, nejlepší byl algoritmus B^3D, který lze spustit na grafické kartě. Dále jsme vyzkoušeli techniku transformace obrazu pomocí Z-křivky a kompresi pomocí rozdílu dvou snímků. Ani jedna z těchto dvou metod, avšak nevedla k lepším výsledkům.

V druhé, hlavní části této diplomové práce jsme se zabývali ztrátovou kompresí obrazu. Kompresní poměry bezztrátové komprese jsou limitovány, proto jsme rozhodli využit právě ztrátové komprese. Ta nám dovoluje dosahovat lepších výsledků za cenu ztráty určité obrazové informace. Nejdříve jsme uvedli standardní existující metodu JPEG, která je velmi rozšířená a její varianty se používají i pro kompresi snímků z mikroskopů. Poté jsme se již dostali k popisu a vysvětlení námi implementovaných kompresních metod. Začali jsme skalární kvantizací, což je jedna z jednodušších metod ztrátové komprese. Zde jsme si vysvětlili proč vůbec funguje a popsali jsme algoritmus pro nalezení optimálního slovníku. Následovala vektorová kvantizace, která je zobecněním skalární kvantizace a využívá komprese více prvků obrazu najednou. Detailně jsme popsali algoritmus LBG, pomocí kterého tvoříme slovník pro kvantizaci. Tento slovník obsahuje co nejlepší vektory pro rekonstrukci obrazu.

Závěrečnou část jsme otevřeli popisem datasetů, na kterých jsme testovali naše metody. Také jsme zde uvedli některé zajímavé vlastnosti bioinformatických datasetů. Následně jsme se již dostali k prezentaci a analýze výsledků, kterých jsme dosáhli. Zjistili jsme, že kvantizační chyba u obrazů komprimovaných pomocí skalární kvantizace je velmi dobrá. Původní kompresní poměr byl velmi limitován a nebyli schopni dosáhnout méně než 8 bitů na pixel. Toto platí v pří-

padě největšího slovníku. Použitím Huffmanova kódování jsme nakonec byly schopni dosáhnout hodnoty cca 6 bitů na pixel, v případě slovníku s 256 skaláry. Při kompresi pomocí vektorové kvantizace jsme dosáhli mnohem lepších kompresních poměrů a pro všechny slovníky jsme byli schopni se dostat pod 1 bit na pixel obrazu. Cenou za tento výborný kompresní poměr byla větší základní chyba. Distribuce hodnot kvantizační chyby byla sice centrována kolem nuly, ale histogram byl více rozšířen do obou stran.

U obou našich metod jsme představili kompresi pomocí globálního slovníku, vytvořeného ze všech rovin daného datasetu. Při využití tohoto typu slovníku byly výsledky komprese stabilnější, hlavně pro menší L. Co je hlavní, průměrná chyba pixelu, kterou jsme vypočetli pro všechny řezy, nebyla horší, než tomu bylo při použití individuálních slovníků. Navíc touto technikou jsme schopni dosáhnout vyšší rychlosti jak komprese, tak i dekomprese. Tyto rychlosti obou metod jsme porovnali a zjistili jsme, že dekomprese při vektorové kvantizace je velmi dobrá. Zároveň se tato rychlost dobře chová i s rostoucím L.

V budoucí práci bychom chtěli zapojit vektorovou kvantizaci do nástroje BigDataViewer. Konkrétně bychom zapojili kompresní algoritmus na stranu serveru, který by posílal komprimovaný proud dat klientovi. Klient by byl schopen pomocí našeho dekompresního algoritmu data rozbalit a zobrazit. Podle námi získaných výsledků, jsou kompresní poměry velice dobré, čas dekomprese je krátký a vzniklá vizuální chyba je minimálně. Tímto myslíme, že chyba v obraze je těžko detekovatelná pro člověka, pozorující dataset na monitoru počítače. Dále bychom chtěli vyzkoušet naše kompresní algoritmy na širším spektru dat. Tím myslíme více různých datasetů. V neposlední řadě bychom rádi vyzkoušeli další techniky komprese, ať už se jedná o metody transformace dat nebo o zcela jiné kompresní algoritmy, například metody vlnkové komprese.

Literatura

1. PIETZSCH, Tobias; SAALFELD, Stephan; PREIBISCH, Stephan; TOMANCAK, Pavel. BigDataViewer: visualization and processing for large image data sets. *Nature methods*. 2015, roč. 12, č. 6, s. 481.
2. SCHINDELIN, Johannes et al. Fiji: an open-source platform for biological-image analysis. *Nature methods*. 2012, roč. 9, č. 7, s. 676.
3. *ZISRAW (CZI) File Format*. 2016. Č. MC33879. Version 1.2.2.
4. *Open Microscopy Environment* [`https://www.openmicroscopy.org/`]. Dne: 23.01.2019.
5. BALAZS, Balint; DESCHAMPS, Joran; ALBERT, Marvin; RIES, Jonas; HUFNAGEL, Lars. A real-time compression library for microscopy images. *bioRxiv*. 2017. Dostupné z eprint: `https://www.biorxiv.org/content/early/2017/07/21/164624.full.pdf`.
6. *GZIP file format specification version 4.3*. 1996-05. Č. rfc1952.
7. *DEFLATE Compressed Data Format Specification version 1.3*. 1996-05. Č. rfc1951.
8. ZIV, J.; LEMPEL, A. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*. 1977-05, roč. 23, č. 3, s. 337–343. ISSN 0018-9448. Dostupné z DOI: `10.1109/TIT.1977.1055714`.
9. PAVLOV, Igor. *7-zip, LZMA*. 2019-03. Dostupné také z: `https://www.7-zip.org/sdk.html`.
10. MENA, Federico. *bzip2*. 2019-03. Dostupné také z: `https://sourceware.org/bzip2/`.
11. NICKOLLS, John; BUCK, Ian; GARLAND, Michael. Scalable parallel programming. In: *2008 IEEE Hot Chips 20 Symposium (HCS)*. 2008, s. 40–53.
12. MORTON, Guy M. A computer oriented geodetic data base and a new technique in file sequencing. 1966.
13. WALLACE, G. K. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*. 1992-02, roč. 38, č. 1, s. xviii–xxxiv. ISSN 1558-4127.
14. SOJKA, Eduard. *Digitální zpracování a analýza obrazů*. 1. vyd. VŠB - Technická univerzita Ostrava, 2000. ISBN 80-7078-746-5.
15. DUFAUX, F.; SULLIVAN, G. J.; EBRAHIMI, T. The JPEG XR image coding standard [Standards in a Nutshell]. *IEEE Signal Processing Magazine*. 2009-11, roč. 26, č. 6, s. 195–204. ISSN 1558-0792. Dostupné z DOI: `10.1109/MSP.2009.934187`.
16. SALOMON, David. *Data Compression: The Complete Reference*. Springer, 2006-12. ISBN 1846286026. Dostupné také z: `https://www.xarg.org/ref/a/1846286026/`.

17. SAYOOD, Khalid (ed.). Preface. In: *Introduction to Data Compression (Third Edition)*. Third Edition. Burlington: Morgan Kaufmann, 2006, s. xvii–xxii. The Morgan Kaufmann Series in Multimedia Information and Systems. ISBN 978-0-12-620862-7. Dostupné z DOI: https://doi.org/10.1016/B978-012620862-7/50000-6.

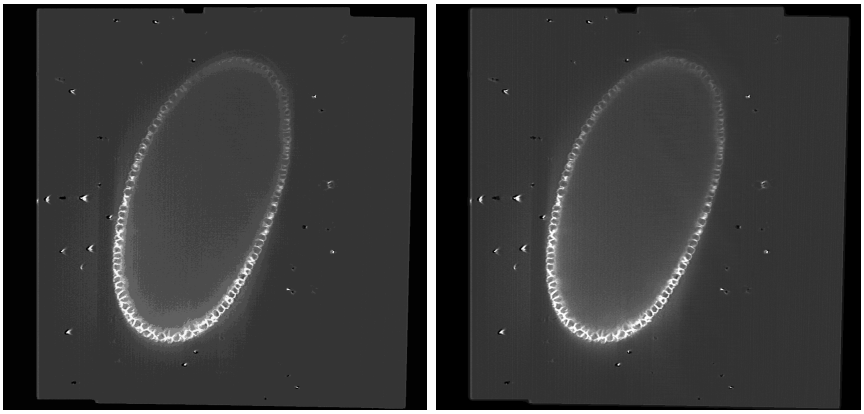
18. LINDE, Y.; BUZO, A.; GRAY, R. An Algorithm for Vector Quantizer Design. *IEEE Transactions on Communications*. 1980-01, roč. 28, č. 1, s. 84–95. ISSN 1558-0857.

19. BENSAID, C.; ABOUTAJDINE, D.; ZYOUTE, M. Adaptive splitting threshold for vector quantization. In: *[1991 Proceedings] 6th Mediterranean Electrotechnical Conference*. 1991-05, 424–427 vol.1. ISSN null. Dostupné z DOI: 10.1109/MELCON.1991.161867.

20. HUFFMAN, David A. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*. 1952, roč. 40, č. 9, s. 1098–1101.

21. HINER, Mark C; RUEDEN, Curtis T; ELICEIRI, Kevin W. SCIFIO: an extensible framework to support scientific image formats. *BMC bioinformatics*. 2016, roč. 17, č. 1, s. 521.

A Výsledky komprese

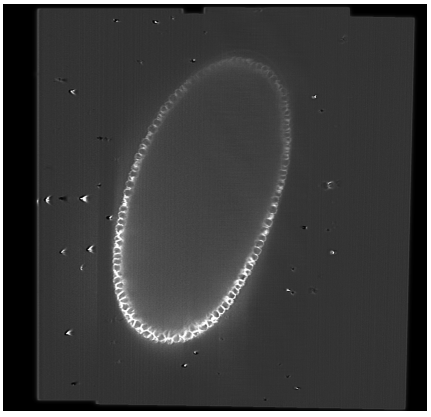


(a) $L = 8$, BPP = 1,9541

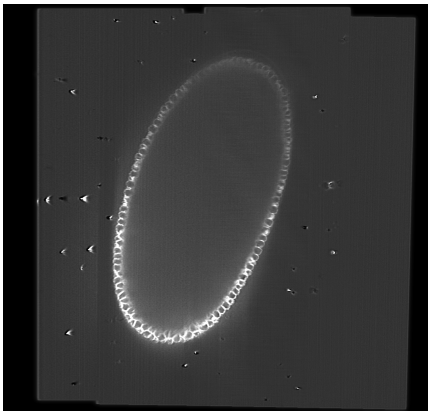
(b) $L = 16$, BPP = 3,0538

(c) $L = 32$, BPP = 3,9268

(d) $L = 64$, BPP = 4,8027

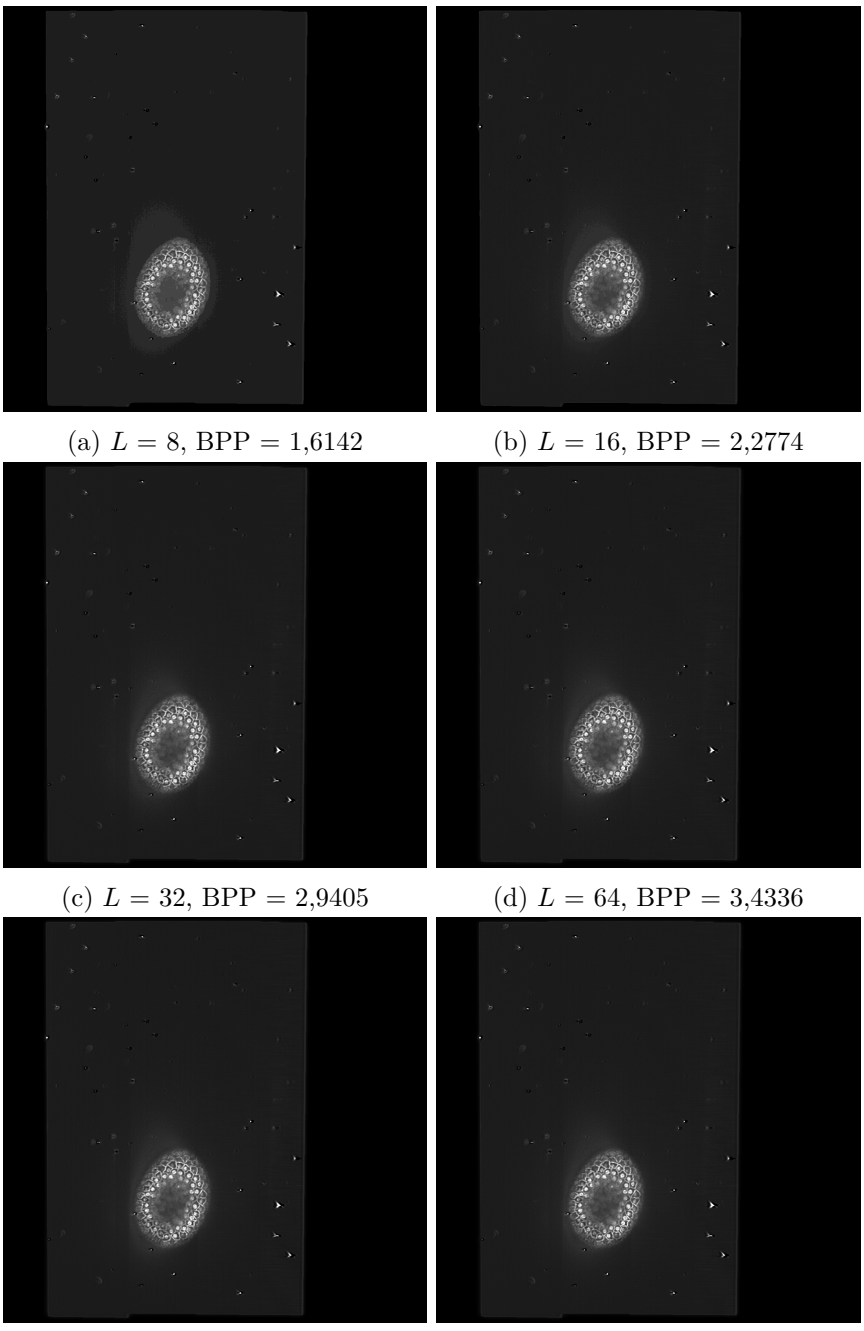


(e) $L = 128$, BPP = 5,2784



(f) $L = 256$, BPP = 5,9502

Obrázek 24: Řez 400 datasetu embrya octomilky (kanál 0) po skalární kvantizaci s využitím Huffmanova kódování

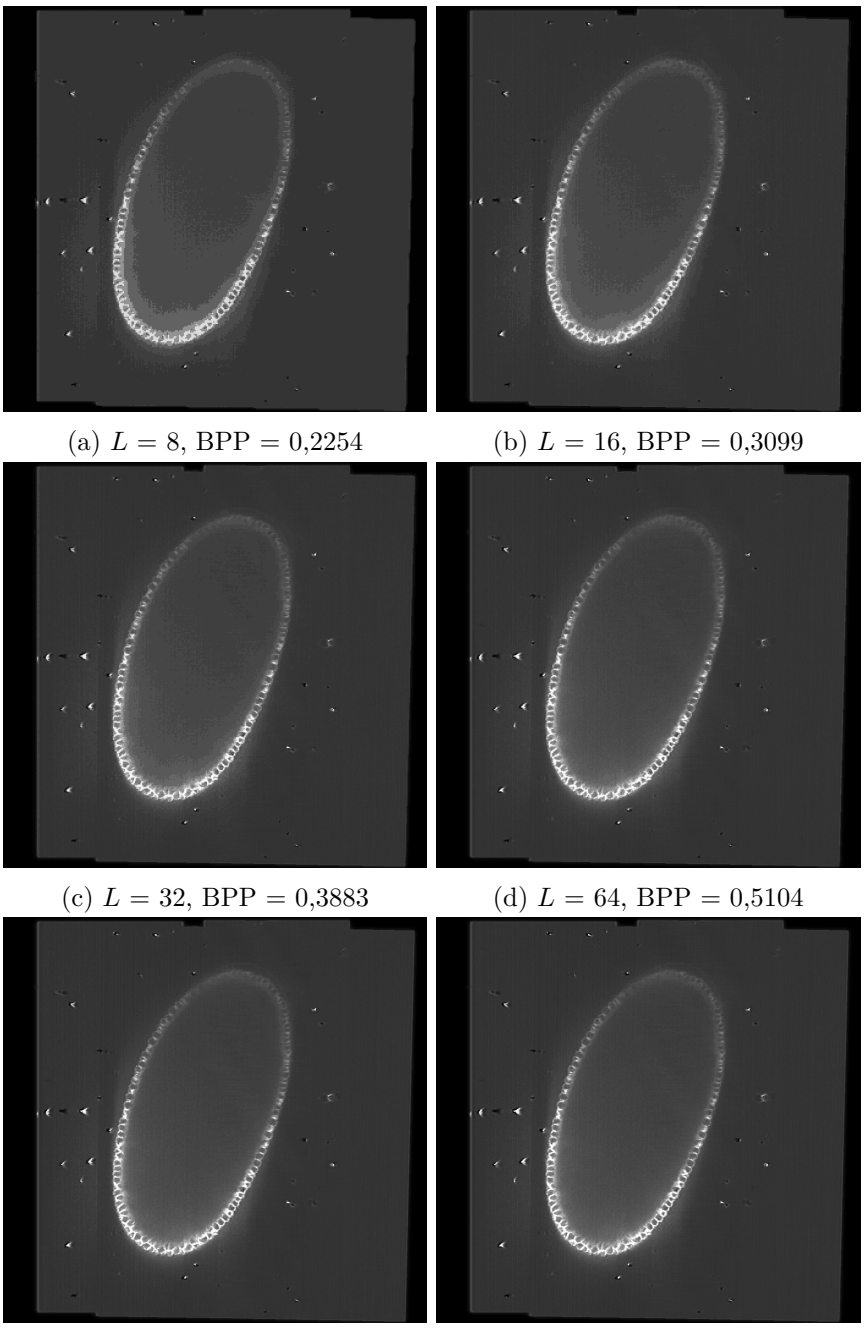


(a) $L = 8$, BPP = 1,6142 (b) $L = 16$, BPP = 2,2774

(c) $L = 32$, BPP = 2,9405 (d) $L = 64$, BPP = 3,4336

(e) $L = 128$, BPP = 3,8471 (f) $L = 256$, BPP = 4,3123

Obrázek 25: Řez 683 datasetu embrya octomilky (kanál 1) po skalární kvantizaci s využitím Huffmanova kódování

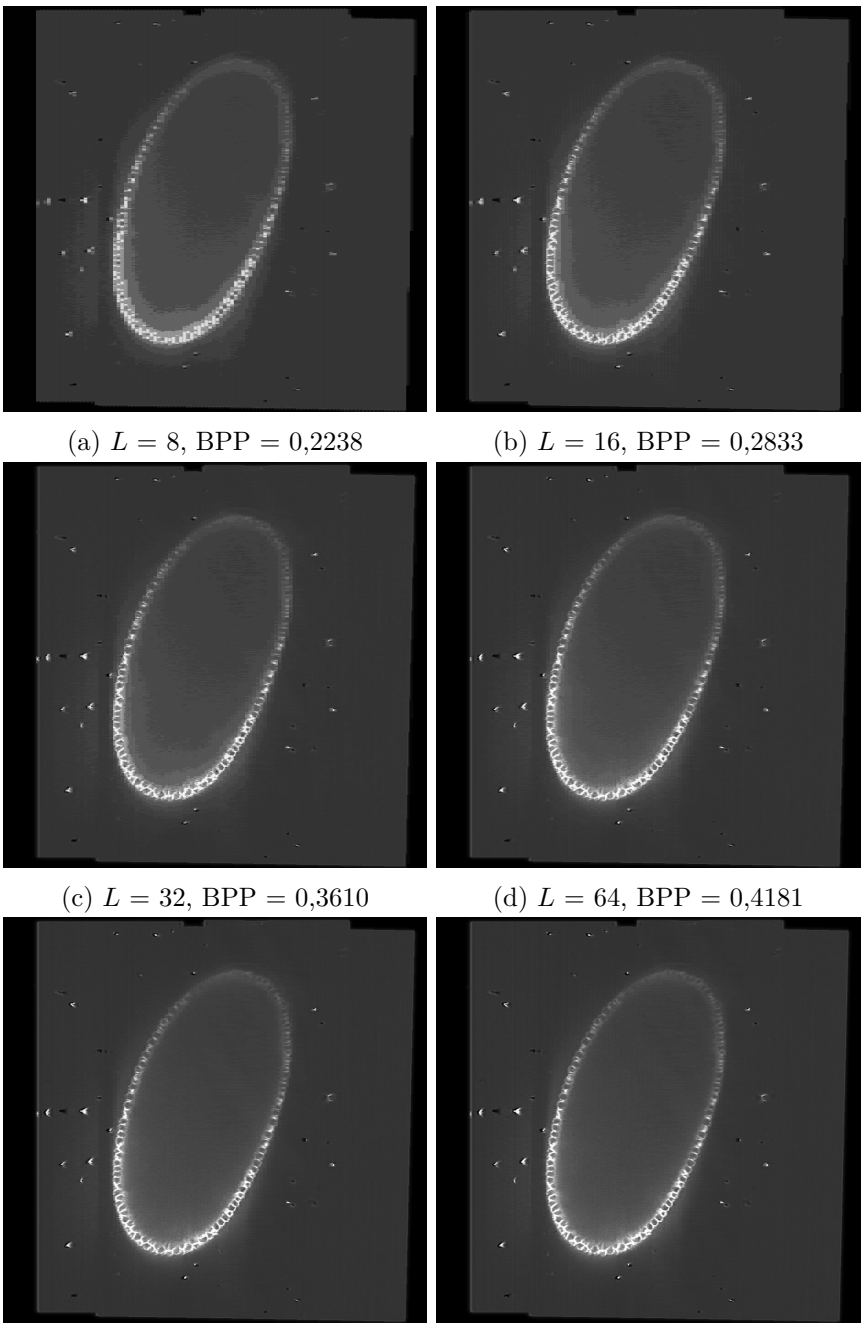


(a) $L = 8$, BPP = 0,2254 (b) $L = 16$, BPP = 0,3099

(c) $L = 32$, BPP = 0,3883 (d) $L = 64$, BPP = 0,5104

(e) $L = 128$, BPP = 0,5957 (f) $L = 256$, BPP = 0,7459

Obrázek 26: Řez 400 datasetu embrya octomilky (kanál 0) po vektorové kvantizaci (3×3) s využitím Huffmanova kódování

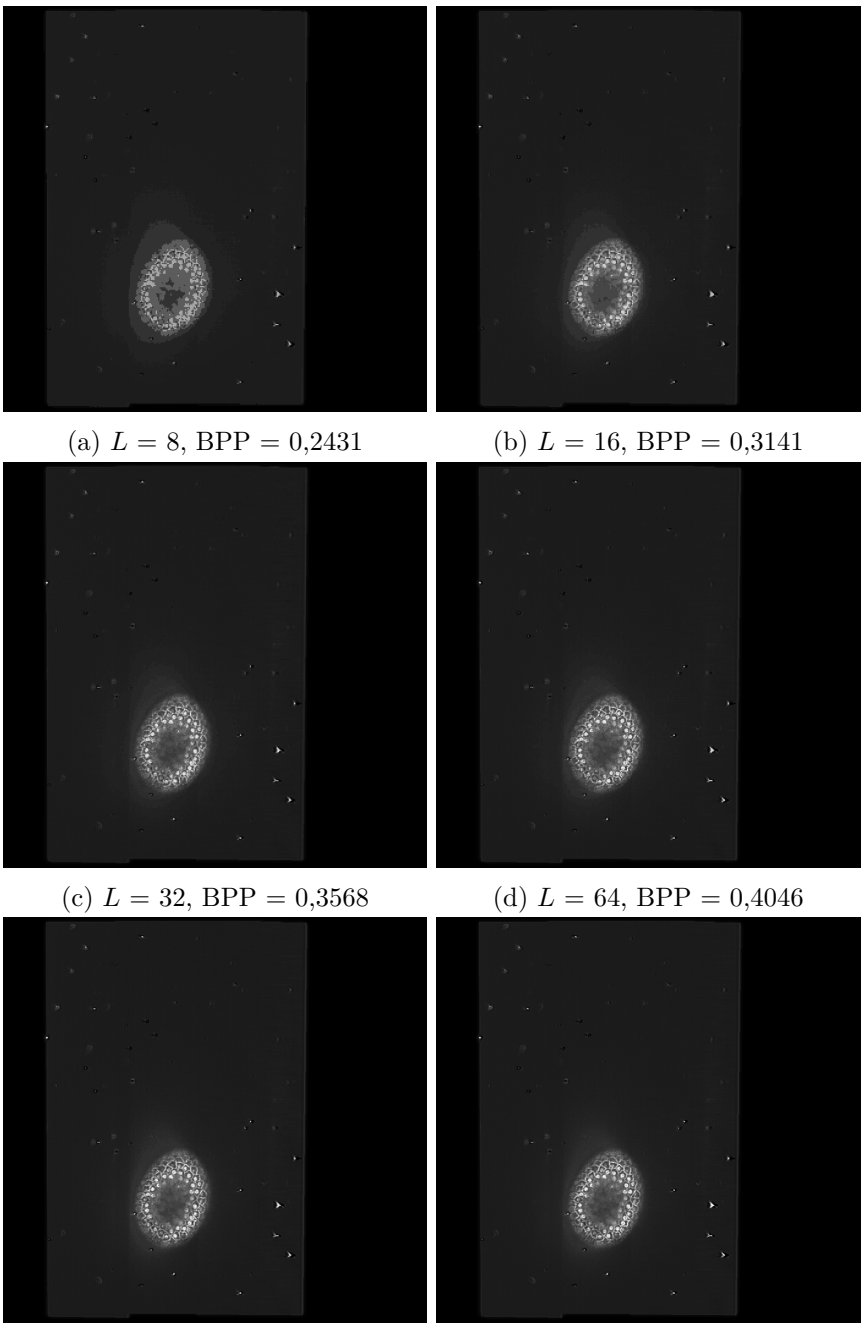


(a) $L = 8$, BPP = 0,2238 (b) $L = 16$, BPP = 0,2833

(c) $L = 32$, BPP = 0,3610 (d) $L = 64$, BPP = 0,4181

(e) $L = 128$, BPP = 0,5472 (f) $L = 256$, BPP = 0,6764

Obrázek 27: Řez 400 datasetu embrya octomilky (kanál 0) po vektorové kvantizaci (9×1) s využitím Huffmanova kódování

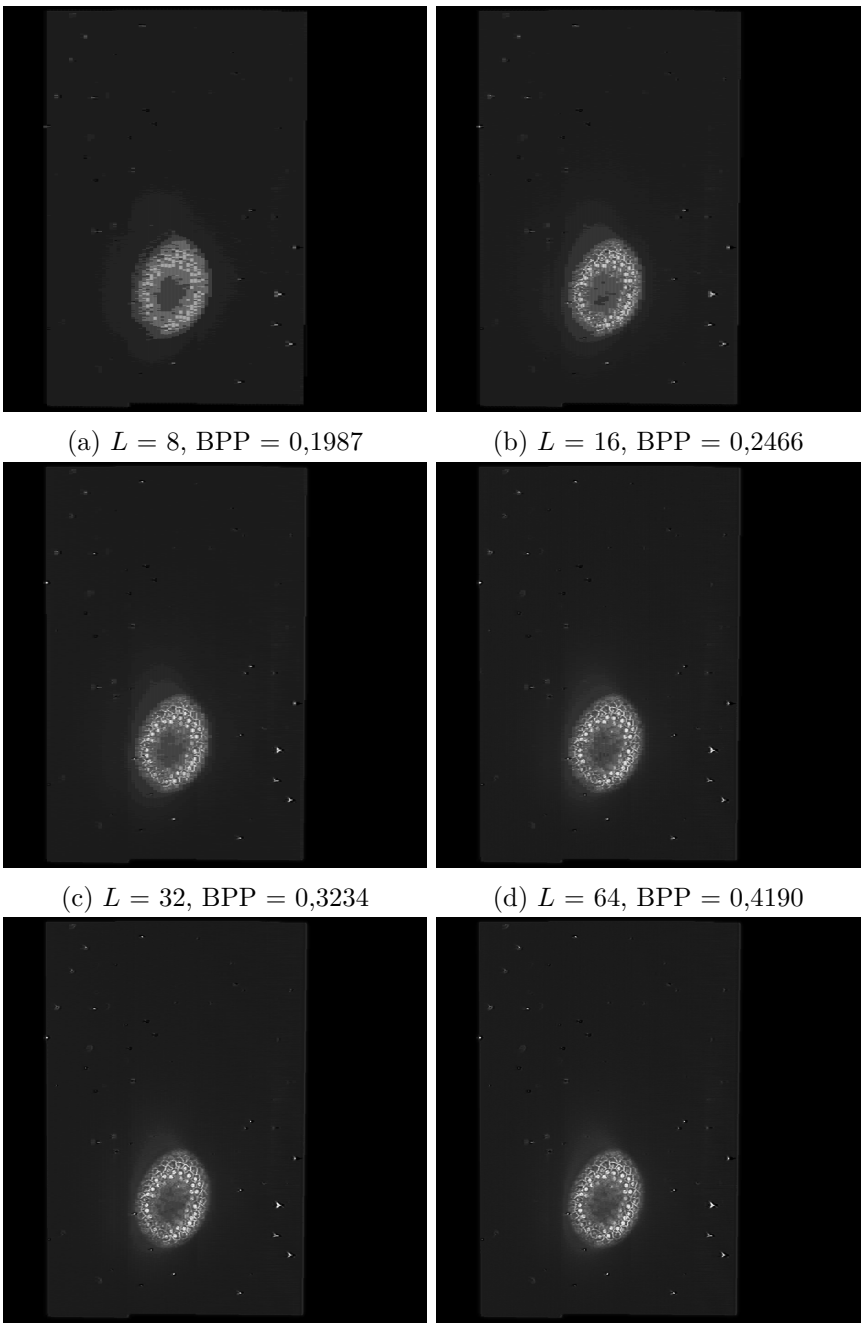


(a) $L = 8$, BPP = 0,2431 (b) $L = 16$, BPP = 0,3141

(c) $L = 32$, BPP = 0,3568 (d) $L = 64$, BPP = 0,4046

(e) $L = 128$, BPP = 0,5040 (f) $L = 256$, BPP = 0,6171

Obrázek 28: Řez 683 datasetu embrya octomilky (kanál 1) po vektorové kvantizaci (3×3) s využitím Huffmanova kódování



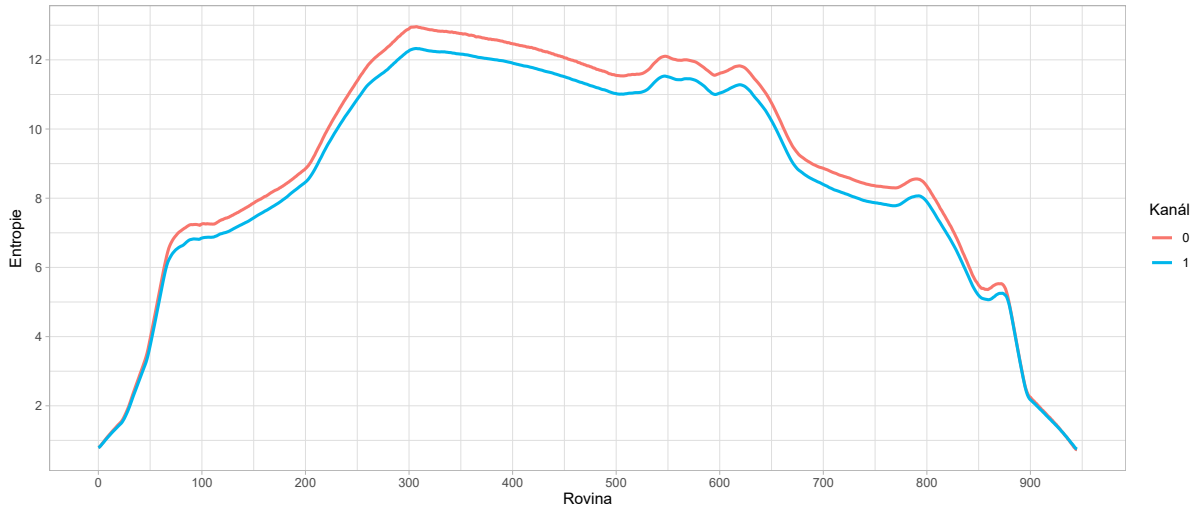
(a) $L = 8$, BPP = 0,1987 (b) $L = 16$, BPP = 0,2466

(c) $L = 32$, BPP = 0,3234 (d) $L = 64$, BPP = 0,4190

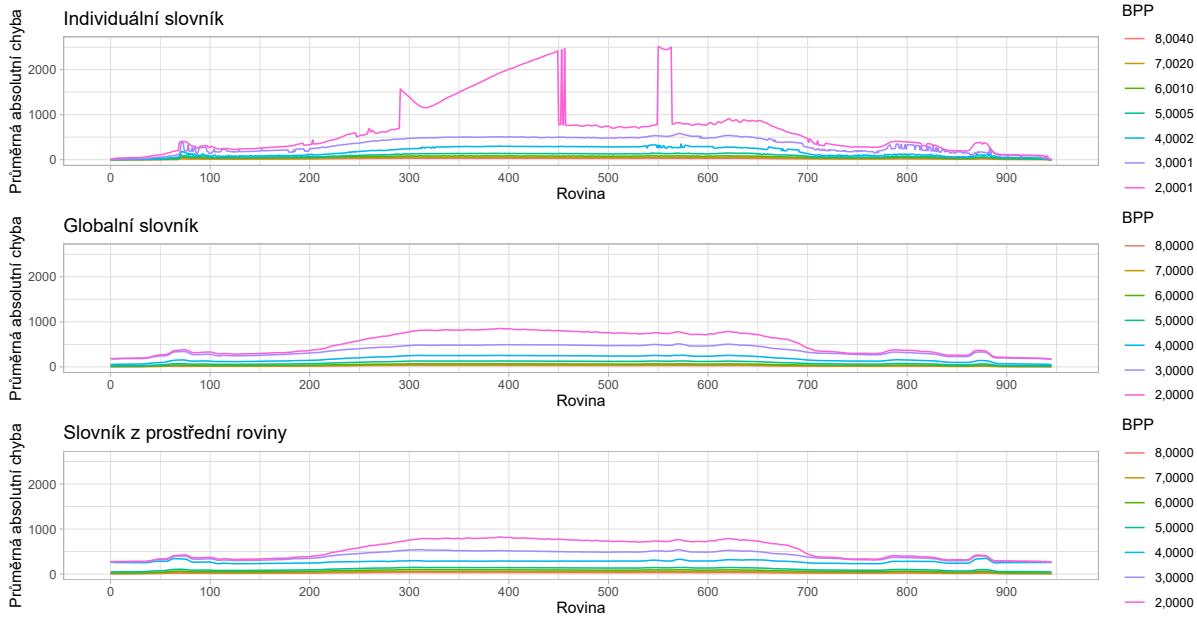
(e) $L = 128$, BPP = 0,4575 (f) $L = 256$, BPP = 0,5739

Obrázek 29: Řez 683 datasetu embrya octomilky (kanál 1) po vektorové kvantizaci (9×1) s využitím Huffmanova kódování

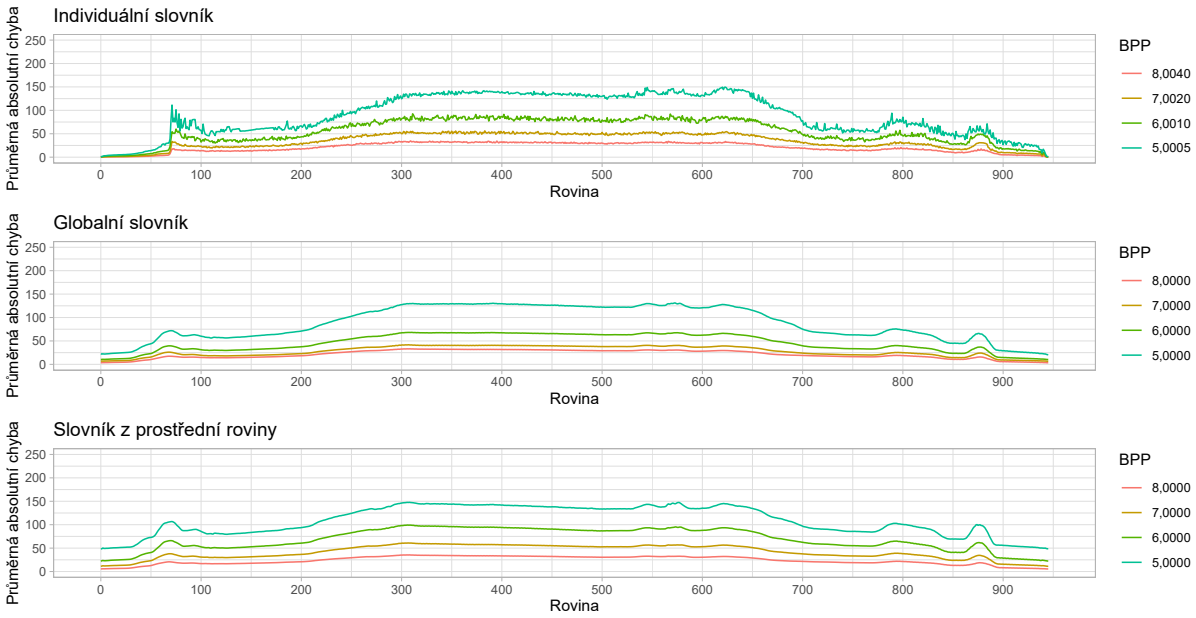
B Doplňující grafy výsledků



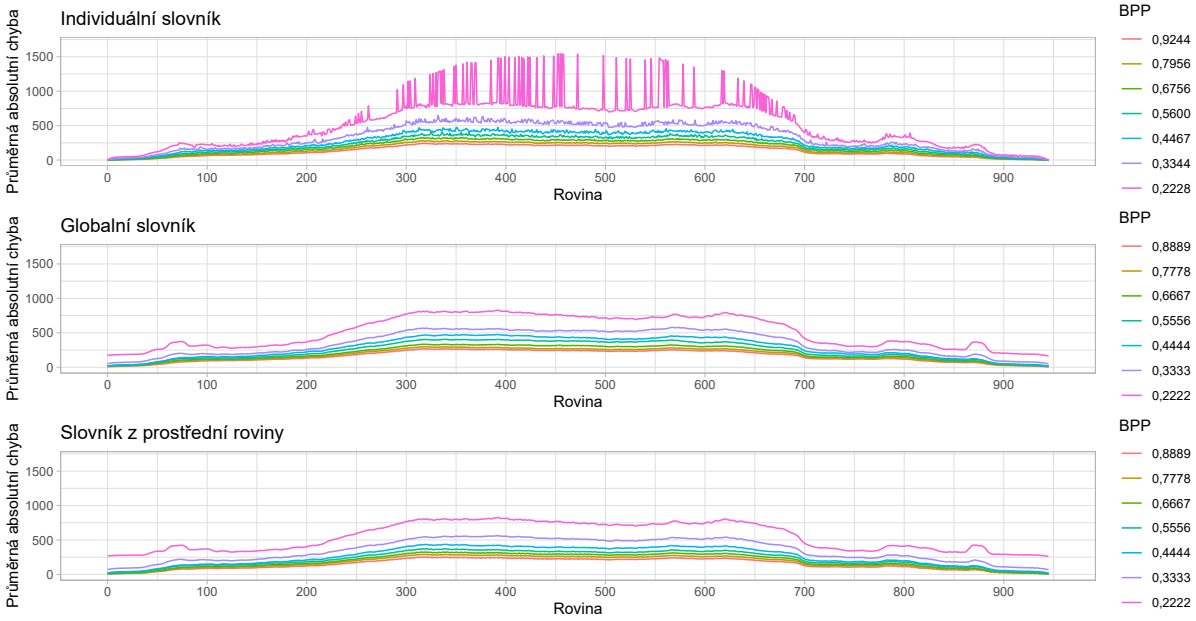
Obrázek 30: Entropie jednotlivých rovin v datasetu



Obrázek 31: Průměrná absolutní chyba skalární kvantizace podle rovin (Kanál 1)



Obrázek 32: Detail průměrné absolutní chyby skalární kvantizace podle rovin (Kanál 0)



Obrázek 33: Průměrná absolutní chyba vektorové kvantizace (3×3) podle rovin (Kanál 1)

C Doplňující tabulky

jp2:rate	MSE	PSNR (dB)	CR	BPP
Kanál 0 řez 400				
1	0,0000	∞	0,7274	11,6384
2	16,5254	84,1479	0,4998	7,9965
4	4747,2703	59,5650	0,2499	3,9979
6	37325,7784	50,6094	0,1665	2,6645
8	81477,9103	47,2191	0,1248	1,9971
10	143942,5801	44,7476	0,0999	1,5990
12	214275,9355	43,0197	0,0831	1,3296
14	267446,5733	42,0571	0,0713	1,1402
16	305597,2032	41,4780	0,0625	0,9998
18	375761,9526	40,5803	0,0554	0,8872
20	472478,8087	39,5856	0,0499	0,7983
22	568262,4498	38,7840	0,0453	0,7249
24	658343,9639	38,1449	0,0416	0,6658
26	755703,5305	37,5460	0,0384	0,6140
28	833956,4025	37,1180	0,0357	0,5709
30	917047,3288	36,7055	0,0333	0,5333
Kanál 1 řez 683				
1	0,0000	∞	0,4961	7,9372
2	0,0000	∞	0,4961	7,9372
4	181,6304	73,7376	0,2500	3,9996
6	2776,6950	61,8942	0,1666	2,6661
8	11006,5322	55,9130	0,1248	1,9965
10	24146,2679	52,5010	0,0999	1,5983
12	36536,0421	50,7023	0,0831	1,3301
14	45829,275	49,7180	0,0713	1,1413
16	54637,9017	48,9545	0,0624	0,9983
18	70514,4769	47,8467	0,0554	0,8858
20	92633,4456	46,6618	0,0500	0,7998
22	117554,9655	45,6271	0,0453	0,7252
24	149517,5475	44,5825	0,0416	0,6649
26	180292,4088	43,7697	0,0384	0,6141
28	256228,3815	42,2432	0,0356	0,5696
30	361035,0555	40,7540	0,0332	0,5320

Tabulka 14: Výsledky komprese pomocí JPEG2000